1-1-1978

Design and implementation of a data base management system application for the small user.

Stephen J. Tillman

DESIGN AND IMPLEMENTATION OF A DATA BASE MANAGEMENT SYSTEM APPLICATION FOR THE SMALL USER

by

Stephen J. Tillman

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1978

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Industrial Engineering.

7-28-78
(date)

Professor Ben L. Wechsler
Major Thesis Advisor

Professor George E. Kane
Chairman, Department of
Industrial Engineering

Acknowledgments

To Dr. Ben L. Wechsler, major thesis advisor, and one of the finest teachers it has been the author's privilege to know and learn from.

To Dr. George C. Driscoll, minor thesis advisor, whose cooperation made working on the case study almost painless.

To Wilkes College, for providing the sabbatical leave and other support necessary to complete the work.

To Irene Cocco, for typing the manuscript.

The author wishes to express his sincere appreciation to the above-mentioned, plus all the others who directly or indirectly, helped make this possible.

TABLE OF CONTENTS

Page

ABSTRACT

This thesis presents general guidelines for the development of a data base management system application. The guidelines are aimed toward a concultant working for a small user group within a large organization which has in-house data base management system software. The guidelines are based upon the author's experience in designing and implementing the membership data base for the "Planning and Design of Tall Buildings" research project. The guidelines are presented according to which phase of the development effort they are best suited for. The categories are the pre-design phase, the schema design, the application programs, documentation and manuals, and implementation. In addition, there is a general category of guidelines, which cut across all phases of the system development.

Along with the guidelines, several areas of future study are presented. These areas are standard topics which arise in many data processing system developments. Because of the pecularities of the case study, they were either not considered, or were considered in such a way that the case study did not offer sufficient insight for generalizing.

CHAPTER 1

Introduction

1.1 General Background

As organizations grow in size and complexity, their informational needs grow with them. In fact, the growth of informational requirements is generally faster than the growth of the organization. As the amount of information increases, more information is needed to keep track of the information already accumulated and being further accumulated. In addition, societal problems, government regulations, and other outside pressures further increase the need for information, data processing, report generation, and so on.

The fantastic growth of computer technology has given organizations the tools necessary to solve, or at least alleviate, their information problem. At the same time it has brought with it another problem--how to make use of the tools so that "the cure does not become worse than the disease." Martin, in [12], says "Already ... about 20% of the U.S. gross National Product is devoted to the collection, processing, and dissemination of information and knowledge ..." Martin is referring to

the total handling of all information, not just computer based information. The fact is, however, that in most medium to large organizations, and in many small ones, computers are the backbone of the information system. Hardware costs, software costs, special personnel costs, the cost of such supporting equipment as punch cards, paper tape, magnetic tape and discs, and the costs of maintaining the physical environment of the computer system are just some of the costs of maintaining a computer based information system. The share of the information expenditures that go toward the computer is a significant fraction of the total information expenditures, and that fraction is growing.

Initially computers in business were used largely as accounting aids. As information needs and computer technology grew, and as the users became more aware of the capabilities and uses to which the new machines could be put, sophisticated applications requiring complex software and large specially structured data files came into being. At first individual applications were treated as though they were largely independent of one another. Each application had its own programs and data files even though there was often a large amount of overlap in both input and output. This situation inevitably led to problems of redundancy, inconsistency, excess

storage and processing costs, expensive time delays, and in general a huge maintenance problem. Some organizations were finding that more than 80% of an exponentially increasing data processing budget was going toward maintaining an increasingly inadequate, and massive, system, and furthermore the maintenance percentage was on the increase. (See [12], p. 46) The sheer size and disorganization of the data processing system made new programming development extremely difficult and costly. In those cases where a new application, with its programs and data files, was successfully developed, the global problem was found to be even worse than before. The new programs and files added to the already overburdened maintenance facilities of the organization. More than one organization folded, at least partially, because it could not solve its information problem.

Some sort of a systematic approach has to be taken to make optimal (or as nearly optimal as possible) use of an organization's informational, and in particular data processing, resources. An organization does not want to limit growth, but the growth should be controlled. Standards have to be set and maintained. One method of maintaining a meaningful set of standards is to build applications around a well designed organizational database contained within a well designed organization wide

information system. Burch and Strater in [4] give an overview of the possible designs of general information systems, and the use of the computer as a major part of the systems. The technology of the 70's has increasingly led toward the use of on-line computer systems, and the data base management systems. Yourdon, in [15], discusses the design of on-line systems in general, including a brief discussion of data base management systems. Martin gives a fairly thorough overview of data base management systems in [12], and goes into details of design considerations and related matters in [11].

Theoretically a data base management system (DBMS) provides one large central data bank. All data for all applications is present in a standard format. The data that a given application needs is easily accessible, but all other data is protected from unauthorized access. Changes in data organization and content do not affect any application program, and application programs may be changed without requiring the data base to be restructured. In practice, unfortunately, this has not come to pass. What has happened, however, is the development of several smaller data bases, designed around specific functional needs and/or common data usage. Each of the smaller data bases serves, in general, several applications each.

The technology necessary for a DBMS is not trivial. Generally organizations which try to design their own DBMS have great difficulties (see case study C p. 388 in Kroenke [10]) or at the very least excess costs. Therefore when an organization makes the decision to implement a DBMS, it usually decides to take advantage of one of the commercial software packages available, either from a computer vendor or from an independent company (see [6] for an independent comparison of several available packages). The packages vary in complexity and cost. An example of a successful, but relatively simple and inexpensive system is TOTAL (see [5]), which is marketed by Cincom Systems, Inc. One of the more complex successful systems is IMS (see [9]), marketed by IBM, but available on some other hardware also. A general guideline for comparison is the CODASYL data base task group (DBTG) system (see [14]), which was designed as a standard for a general purpose DBMS. Many information systems specialists (see [12], p. 148) debate the effectiveness of using the CODASYL DBTG system as a standard, but at the very least it does provide a common benchmark against which other systems can be measured.

Once an organization has made the decision to implement a DBMS, there remains the problem of getting the users to at least consider it in performing their

applications. The implementation decision will have been made, presumably, with several specific applications in mind. The users involved with those applications would probably have little choice about whether or not to make use of the company DBMS, and in any case they probably would want to use it. Within most organizations, however, there are many diverse and independent computer applications. The users connected with applications that are largely independent of the initial DBMS applications could choose to avoid the in-house DBMS, at least for a while. Granted that a DBMS is not always the optimal choice, one has to at least consider a DBMS application before one is in a position to make an intelligent decision.

Even if it is assumed that the design personnel of an organization's EDP department are completely familiar with and know how to make best use of a DBMS (not always a valid assumption), the users are generally unsophisticated in terms of the capabilities and ease of applicability of a DBMS. While they may be willing to believe (because someone told them so) that the organization as a whole is better off with a DBMS, they are often not able to see how it can be applied in their own case. In many cases there is also, quite understandably, the fear of the unknown. The users have to be taught not

only that a DBMS application can be of use to them, but that the application can be carried through without undue strain and mystification, and they will be pleased with the eventual results.

1.2 Statement of the Problem

The assumption here is that an organizational decision to install a DBMS has already been made, and that the installation has been all, or nearly all, completed. In that context, consider the case of a small user group within a large organization that has an in-house DBMS. Assume the user group has need of a data base reorganization for its own activities, and would like to take advantage of the organization's DBMS, or at least to consider the possibility of a DBMS application. The purpose of this thesis is to present a mechanism by which a consultant can aid the unsophisticated user (in terms of DBMS knowledge) in considering and implementing a DBMS application when it is called for. The idea is that the user should be on top of and participate in the entire development effort from inception to implementation, and be very comfortable with the end result. In the foregoing context, the problem to be considered is restricted to that mentioned above. The more fundamental problem of how an organization should structure its data processing

resources is beyond the scope of this thesis, and has, in any case, been treated fairly extensively in the literature (see references cited in 1.1).

1.3 Method of Solution

While it is likely that there is no fixed "all purpose" solution to the problem stated in 1.2, it is also likely that there are some underlying principles involved which would be valid for the vast majority of cases of the type considered here. Therefore the approach taken in this thesis is to present a case study in some detail. Hopefully, the case study will be "typical" enough of such cases to bring out those underlying principles.

To fulfill the purpose of this thesis the reorganization of the membership data base of the "Planning and Design of Tall Buildings" research project will be used as a case study. As will be seen in Chapter II, it is a relatively simple case, but that means that the peculiarities of the case study itself are less likely to obscure the general guidelines involved. It is realized that one case study cannot possibly provide a general guideline that will work in every case. Also, as will be shown later, there are some areas which were not of great concern in this case study, but which can be in other

cases. This thesis is intended to be one step in the overall path toward a general solution.

1.4 Objectives

The objective of this thesis is to provide a guideline which can be used by a small user in the environment mentioned in section 1.2.

CHAPTER 2

The Case Study

2.1 Case Study Environment

The case study took place at Lehigh University. The computer center at Lehigh operates in an "open shop" environment, rather than the more typical "closed shop" environment found in most organizations. The main difference in this particular case is that the consultant (in this case the author) is external to the computer center, rather than internal to it. The hardware is the Digital Equipment Corporation's DEC-20 system. The DBMS software package is DEC's in-house DBMS. This package is appropriate for a case study since it is modeled on, and is very close to, the CODASYL DBTG system.

"Planning and Design of Tall Buildings" is one of several projects under way at the Fritz Engineering Laboratory at Lehigh University. The Fritz Engineering Laboratory is in turn connected with the Civil Engineering Department at Lehigh. The Tall Buildings Project is a large international interdisciplinary research project, centered at Lehigh, and under the direction of

Dr. Lynn S. Beedle. Dr. Beedle, a professor of Civil Engineering at Lehigh, is the director of Fritz Engineering Laboratory. Dr. George C. Driscoll, also a professor of Civil Engineering at Lehigh, is an associate director of Fritz Engineering Laboratory. Among other things, Dr. Driscoll is responsible for heading the computer systems and operations phase of the Tall Buildings Project.

The Council on Tall Buildings and Urban Habitat was established to study all aspects of the planning, design, construction, and operation of tall buildings. One of its major tasks is to come out with a comprehensive MONOGRAPH on the subject. (For a more detailed overview of the Tall Buildings project and its various phases and operations, see Beedle [1] and [2], and Brinker [3].) Thousands of people all over the world are connected with the project in scores of different activities. Needless to say, coordinating their efforts is a major undertaking, and would be extremely difficult without a good computerized membership data base.

Basically, the membership data base is supposed to aid in keeping track of who is doing what, where, and when. In addition, the data base is supposed to provide input for a series of application programs which provide lists of certain project members, in some cases with

addresses, by project activity and/or by organization and/or by geographic location. One of the programs prints address labels for mailing material to selected collections of project members.

With the installation of the DEC-20 system, along with its DBMS software package, at Lehigh in the fall of 1977, Dr. Driscoll saw a way of restructuring the membership data base to avoid then current problems with updating, inconsistent data, redundancy, and in general the usual problems an organization has which makes it turn to a DBMS system. (A brief description of the old system is contained in 2.2. For a fuller description, see Brinker [3], appendix F.) Therefore, after consultation with Dr. Ben L. Wechsler of the Industrial Engineering Department at Lehigh, Dr. Driscoll and Dr. Beedle offered the author the position of research assistant to work with Dr. Driscoll in the design and implementation of the DBMS application which makes up this case study.

2.2 The Old System

When an individual member of the Tall Buildings Project is engaged in a specific activity, the name of the member is placed on the list of those involved with that particular activity. The major lists, for data base purposes, are referred to as rosters. Many of the

rosters are subdivided into sublists, referred to as committees. A few of the rosters are divided into sub-collections, referred to as groups, and the groups are further subdivided into committees. To distinguish between the two types of committees, when that is necessary, they will be referred to as roster committees and group committees. This is a vast oversimplification of the organizational structure of the Tall Buildings Project, but for data base purposes it will suffice.

The old data base was kept on tape as a large sequential file. All processing and applications were performed using Lehigh's Control Data Corporation model 6400 computer system. The exact details of the file organization and how the application programs were run are in [3]. A brief description will be included here for convenience. Each physical record in the tape file was an 80 character card image record. Basically the file can be thought of as being in four parts, separated by specific records used as delimiters.

The first part was a list of all the rosters, groups, and committees. If a roster was broken up into groups, the roster's record was followed by the record of one of the groups in the roster, which was followed by the records of all the group committees within that group. This pattern was repeated until all the groups

within the roster had been listed. If a roster was broken up into roster committees, the roster's record was followed by all the committee records of the committees within the roster. For example, suppose there were five rosters. Suppose rosters 1 and 4 had neither groups nor committees, rosters 2 and 5 had 2 and 4 committees respectively, and roster 3 had 3 groups with 1, 2, and 3 committees respectively. Then a logical picture of the organization of the first part of the file would be as shown in figure 2-1.

The second part of the file contained the actual membership data. Four of the 80 character records were used for each member. The first record contained the first line of the member's mailing address (which would be the title and name in most cases), followed by the member's last name, title, and initials (if the member was a person, as opposed to an organization, in which case just the name of the organization would be included). The second record contained the second line of the mailing address, followed by the member's organization. The third line contained the third line of the mailing address, followed by a code giving the rosters, groups, and/or committees the member was currently on. The fourth record

roster 1 data

roster 2 data

committee 2-1 data

committee 2-2 data

roster 3 data

group 3-1 data

committee 3-1-1 data

group 3-2 data

committee 3-2-1 data

committee 3-2-2 data

group 3-3 data

committee 3-3-1 data

committee 3-3-2 data

committee 3-3-3 data

roster 4 data

roster 5 data

committee 5-1 data

committee 5-2 data

committee 5-3 data

committee 5-4 data

figure 2-1

contained the fourth line of the mailing address, followed by the member's city and state or country.

The third part of the file repeated the listing of the rosters, groups, and committees in the same order as in the first part. In this case each such entity was followed by a list of the members connected with the entity. Every time a member was connected with a particular entity, the first record of the member's four record data description (described in the previous paragraph) would be duplicated after the record of the entity.

The fourth part of the file contained a list of all the countries that members of the project lived in. Each record contained the name of a country, and its abbreviation.

Each member of the project was on at least one roster, group, or committee, and some were on close to twenty. A change to any single record in the tape file required a separate user input. Thus if a member's name were changed, several different records had to be separately updated to reflect a change in one item. If an organization had twenty project members working for it, the name of that organization was repeated at least twenty times. If the organization changed its name, all those records had to be separately updated. The

same repetition held with cities, states, and countries.

As is the case with tape files, the only way to update the file was to recopy the entire file, changing those items that required change along the way. Updating the file was both difficult and prone to errors that were difficult to correct. As data for file updates accumulated, it was punched on cards and saved. About every three months the entire file was updated using a system software utility program called UPDATE.

The application programs were generally used to create lists of selected member names, sometimes with addresses, and/or to print mailing address labels for selected subsets of the membership. The selected subsets were usually the members associated with particular rosters, groups, and/or committees, not necessarily the same ones every time. The lists were to be sorted alphabetically by name and/or by country, city, and name and/or by organization and name. (States in the United States were treated as countries for this purpose.) If a city, state, country, or organization was entered incorrectly during an update run, the desired lists in subsequent application runs would be messed up. When an application program was to be run, UPDATE was used, along with the accumulated corrections (which were not actually entered into the data base unless the run

coincided with a quarterly file re-creation), to create a temporary disc file consisting of a copy of the relevant records of the data base. After the application was run, the temporary file was discarded.

There were no provisions for querying the data base if information was required about a few individual members. There were also no provisions for keeping such occasionally needed data as telephone numbers and length of time with the project. In general, the old data base was cumbersome to use at best, did not take advantage of advanced computer technology, and as inconsistencies developed, was in danger of becoming a liability.

2.3 Case Study Objectives

Most of the problems with the old system had to do with faulty and delayed updating. Also the file organization made new application development difficult, if not impossible. The primary objective in this case was to have a file organization that makes updating fast and easy, and that does away with internal inconsistencies and redundancies. If possible, the users wanted to have on-line updating facilities. As a spin-off of the on-line updating facilities, it became possible to add the additional objective of on-line querying, at almost no charge. This was possible because querying

would impose no additional data structure constraints, and could be accomplished with a relatively simple application program, as will be seen in 2.7. Another important objective was to allow the same "batch type" applications as were run under the old system. In fact, if possible, the users wanted to use versions of the same application programs. This latter point was because it will be the users' responsibility to maintain the system, and maintenance would be easier if the users were familiar with the application programs. This will be discussed more fully in 2.7.

On a more long range basis, an objective was to provide a data base design which could be easily expanded to include other projects in Fritz Lab. Another objective along the same lines was to have a design which could be emulated for other uses, e.g. an administrative data base at Fritz Lab.

2.4 DBMS Application Decision

It should be clear from the objectives of the case study that the file organization of the data base should be one that allows random access. A given member of the project is on a membership list, a geographical list, an organizational list, and at least one and perhaps as many as twenty roster, committee, and/or group lists. These

factors plus the desired on-line features made the choice of a DBMS application a natural one. Most DBMS software is designed to be compatible with on-line applications. Such software was available. The software would take care of the mechanics of setting up the necessary chains to correspond to all the desired lists. The random access feature of the DBMS was better than any of the available alternatives on the current Lehigh hardware configurations. Finally, a DBMS design makes file maintenance easier from the users' point of view. Frequent file reorganizations are not nearly as likely to occur as with indexed sequential or straight random access methods. This last consideration is especially important in cases where the users are not primarily data processing oriented, but are still responsible for file organization and maintenance.

There were some drawbacks to the decision to go DBMS. First there is the obvious one that the data had to be converted from use on the CDC 6400 to use by the DEC-20. The two systems are almost totally incompatible. This caused more problems than was originally apparent. The CDC used only seven-track tape, so the original data was naturally stored on that medium. The DEC had only one tape drive, and it took nine-track tape. While the computer center claimed they were ready, willing, and able

to help users convert, their cooperation in this particular endeavor was difficult to obtain and its lack unnecessarily delayed completion of the system conversion.

Another drawback to going DBMS was that the applications were largely batch processing oriented, and required use of a high speed line printer. The DEC-20 configuration had only one 240 line-per-minute printer. Eventually some compromises were called for. The project leaders agreed to use the printer for only the first copy of lists and mailing labels, and use copying facilities for others.

A final problem was that the DBMS software was new and still largely experimental. As a consequence, it was not completely debugged. This led to some problems which were overcome by some minor design changes and a different approach toward the application programs.

2.5 Getting Started

Throughout the course of the case study, the users, represented for the most part by Dr. Driscoll, and the consultant worked closely and harmoniously together. This cannot be overemphasized. While in general user involvement is extremely important in the development of an information system, in this type of situation it is even more crucial (if that is possible). Not only will the users

have to use the system, they will have primary responsibility for maintaining it. Also, in this case, the batch application programs are primarily the users'. Therefore, Dr. Driscoll was actively involved in the technical end of the development.

After the initial procedures of engaging the consultant, defining the general scope of the problem, and deciding to go DBMS were completed, Dr. Driscoll and the consultant jointly came up with two documents for guidance. The first was a general schedule (see figure 2-2). No timetables were set, as the duration of each activity was not easily determined in advance. Estimates would have been useless, as neither Dr. Driscoll nor the consultant were able to devote full time, or even a steady percentage of time, to the case. The purposes of the general schedule were to inform everyone involved in the development effort of the general progress made, and to avoid going off on tangents.

The second document was a written description of the data base functions (see figure 2-3). The purpose for this was to have the specific goals in mind and clearly spelled out while doing the work. Both the users and the consultant were aware that this was a working paper only, subject to change if situations warranted (which they did). Both documents were the

General Plan for Tall Building Project Data Base

I. Get functions of data base defined by user

II. Design data base

III. Review design with user, make changes as necessary

IV. Get data base schema up on machine

V. Load test data

VI. Review application needs with user

VII. Write and test application programs

VIII. Write documentation for application programs

IX. Write user manuals, review use with user

X. Load real data

XI. Retest application programs

XII. If time permits, add additional features as requested

figure 2-2

Data Base Functions

I. For each member, the data base will contain his last name, first name (or initial), middle name (or initial), mailing address, telephone number, starting and ending date with the project, geographical data, organizational data, and roster, group, and/or committee affiliations, together with special responsibility where applicable.

II. The major use will be to print names and/or addresses and/or address labels in any of the following orders:

   A. Alphabetically

   B. Chronologically

   C. By country and city, alphabetically

   D. By roster alphabetically. In this case if a name is on several rosters, the user will have a choice as to whether or not more than one address label will be printed

   E. By organization, alphabetically

   F. By roster by starting date in the project

III. Updates to the data base can be made on-line as they occur. Queries of the data base can be made on-line.

figure 2-3

final result of several meetings of give and take.

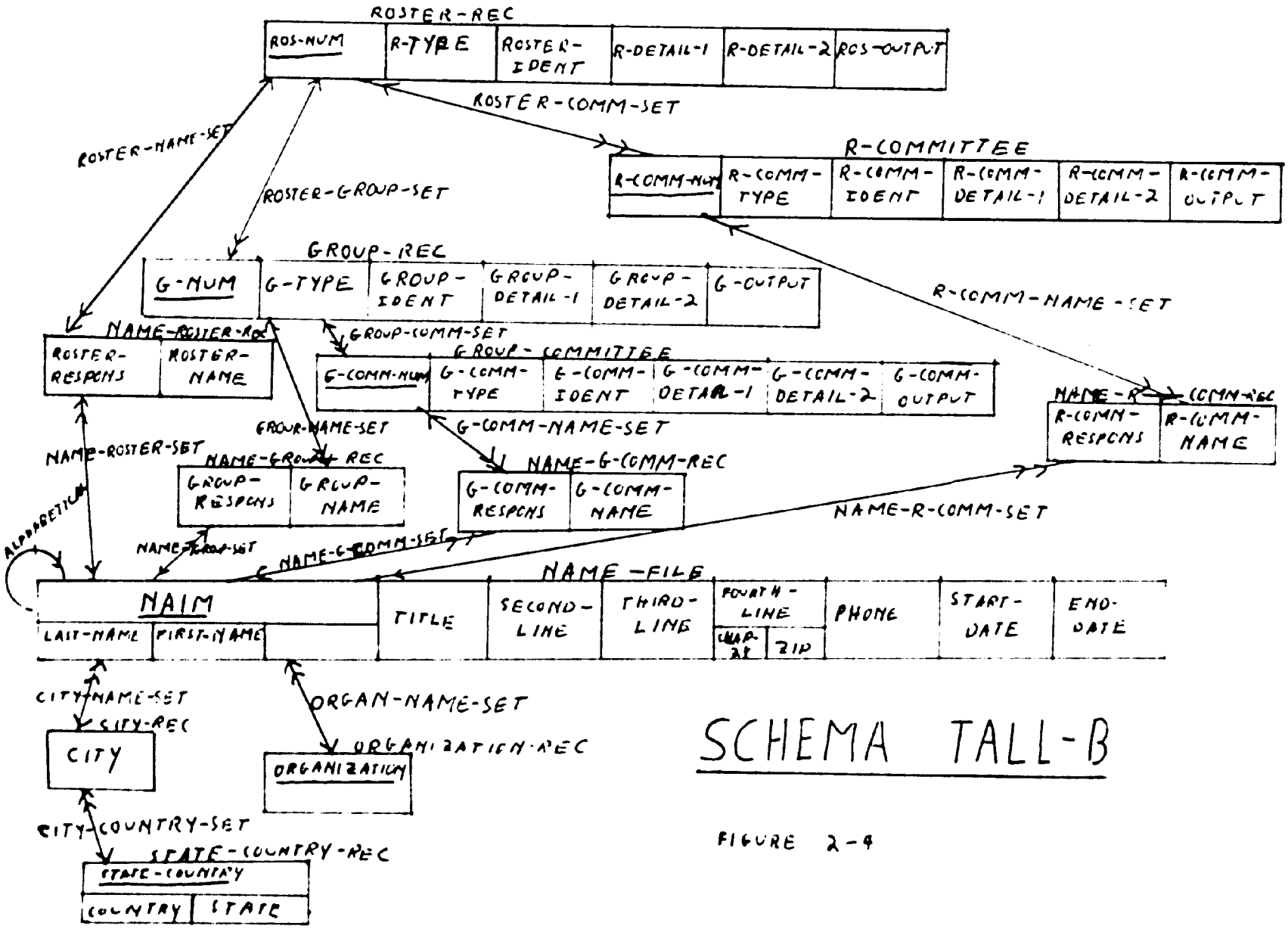
2.6 The Data Base Design

The logical data base design, or schema, is shown in figure 2-4. The actual data description is in appendix A. It is written in the DEC Data Description Language (DDL) (see [7]). The data base dictionary defining the terms used in the design is in appendix B. The size of the data base was worked out by the consultant after consultation with Dr. Driscoll about the number of records of each type. Space was left for growth within each data base area, and pages were left for expansion of each data base area if necessary.

Naturally the design shown in figure 2-4 was not the original one. Some of the changes that were made, along with the reasons for those changes are as follows:

a. Neither groups nor group committees were considered in the original design. They were overlooked when the organizational structure of the project was first described to the consultant. When they, along with their member lists, were added, several new record types and sets had to be added, but the basic design structure remained intact.

b. At first member names were not included in record types NAME-ROSTER-REC, NAME-R-COMM-REC, NAME-GROUP-



SCHEMA TALL-B

FIGURE 2-4

REC, and NAME-G-COMM-REC. It was originally felt that owner pointers to NAME-FILE would be enough, but later the names were added to ease and speed up processing. This is an example of processing speed versus redundancy trade-off, leading to a case of controlled redundancy. Although member names are duplicated, sometimes many times, the application update program is written so that a change in name need only be entered once. Then the name change will be made on the member's personal data record and all appropriate roster, group and committee lists in a manner that is transparent to the user.

c. The NAME-FILE record type was originally ordered two ways--alphabetically and chronologically. Also the ORGANIZATION-REC and STATE-COUNTRY-REC record types were ordered alphabetically. All these orderings, except the alphabetical ordering of NAME-FILE, were eliminated. The main reason was that on test data the system software had a great deal of trouble with system sets. (Often the software would insert a record into the data base correctly, and then would crash while trying to make the appropriate system set linkages. Some of the linkages would get made, but not all. The selection of which linkages would be fouled up, if any were, seemed to be random. The problem went away when the data base design did away with most of the system sets.) Also, a

change in how the batch application programs would be run, which will be described later, made all but the alphabetical ordering of NAME-FILE largely unnecessary.

The software problem was described to the computer center, but no explanation was ever received. Two communications from the center were received. The first said that the problem was being looked into, and may have to be sent to DEC. The second said that the data base (one of the test versions) had been destroyed, and to let them know if the problem reoccurred.

d. Owner pointers were not included in all sets at first, but they were later added to ease processing.

e. The data item FOURTH-LINE was not originally considered necessary, as the consultant felt it could be generated from the city and state or country. Here is an example of the importance of user participation. Dr. Driscoll was able to point out that the fourth line on an address label is restricted in length, and that the actual fourth line might be in a different form than city and state or country. Also it would be easier for the user during an update run to be able to enter the entire address, and not worry about which part to enter and which part not to enter. In this case the major trade-off was user convenience versus redundancy.

f. The internal reference numbers of the rosters,

groups, and committees, i.e. the data items ROS-NUM, G-NUM, R-COMM-NUM, and G-COMM-NUM, were added after the original design was made. They were added because the users wanted to have a simple unique way of identifying each such entity. Then later on, the output codes, i.e. the data items ROS-OUTPUT, G-OUTPUT, R-COMM-OUTPUT, and G-COMM-OUTPUT, were added because they could supply a mnemonic code on printed outputs to identify the rosters, groups, and committees.

2.7 Application Programs

There are actually two types of application programs. There are the users' application programs to handle the users batch programming needs, and the data base programs to handle communication with the data base. The first type can be created to use the data base directly for input, but for reasons which will be discussed in section 2.7.3, that option was not chosen in this case. Therefore, the users' batch application programs were not actually part of the consultant's work, and so will not be discussed here, except as they relate to the DBMS application.

There were four major DBMS application programs written by the consultant--a loading program to load the initial data into the data base, an update program to

allow on-line updates of the data base, a query program to allow on-line queries of the data base, and an interface program to allow the data base to interface with the users' batch application programs. All of these programs were written in COBOL, using the DEC's Data Management Language (DML) verbs, which are COBOL compatible (see [8]).

2.7.1 The Loading Program

The flowchart and written documentation of the loading program are contained in appendix C. Of the four programs, the loading program was the only one written for limited use. It is also the only one that requires no interactive input. Thus no user manual was written for use with this program. On the other hand, the loading program was the only one that required direct user participation in the logical design. The users had to supply the input files for the program. Therefore, the logical flow of the program depended upon what the users gave the consultant to work with.

The input files came from the old data file, which had to be converted from seven track tape to nine track tape. As mentioned in 2.4, there were some problems in getting the computer center to get moving on the conversion. Consideration was even given to using the update

program to load the data base, though that would have been excruciatingly slow. Fortunately this was not necessary as the computer center was able to provide the necessary data files.

The raw data, however, was not what Dr. Driscoll wanted the loading program to use. The data was incomplete. Dr. Driscoll wanted to edit the data to correct some obvious mistakes, and to insert such items as the internal reference numbers and the output codes for the rosters, groups, and committees. They were crucial for the successful running of both the batch and the data base application programs. He also wanted to insert an indicator to differentiate people members from organizational members. Other items which were not in the original data file, such as the first and middle names of person members, the phone numbers of the members, and the starting dates and ending dates of members could be added piecemeal using the update program.

Using a test data file similar in format to the real one, Dr. Driscoll created several different types of new data files with programs that he wrote. Some of the new files were the final product of a series of programs. After several hours of discussion, which took place over a period of a few weeks, Dr. Driscoll and the consultant agreed that the data for loading the data base would be on

two files that were created by Dr. Driscoll from the original data file.

The first file was a list of the rosters, groups, and committees in the same order as in part 1 of the original file (see figure 2-1). In addition to the descriptive data, each record in the file contained the internal reference number and the output code of the particular roster, group or committee that the record was for.

The second file was a list of the project membership. For a person member, the record contained the member's last name and first and middle initials. For an organizational member, the record contained the entire name (up to a certain number of characters). In addition, each record contained a member's four line mailing address, the member's city and state or country (this was separate from the mailing address), the member's organization, plus some other data from the old file not used by the loading program or the data base. The above data came from the old file. Also in each record was a character to indicate whether the member was a person or an organization, and an item giving the total number of rosters, groups, and committees that the member was on, together with a list (by internal reference number) of the appropriate entities with the member's responsibility on each such entity.

These last items were inserted by Dr. Driscoll's programs.

2.7.2 The Update and Query Programs

The flowchart and written documentation for the update program are contained in appendix D, and for the query program in appendix E. User manuals for the two programs are in appendix G. These two programs had the least user involvement in their design and functions. Basically, the users had to indicate satisfaction, or lack thereof, in the end results. There was little need for discussion about what the programs had to do, as their functions were obvious. In both cases the users' primary responsibility was to indicate whether or not they could use the programs satisfactorily, and whether there were any areas requiring improvement.

More time was spent on the update program than any other single feature of the consultant's work. For several weeks nothing else was done, and after that a good percentage of the consultant's time was spent on testing, debugging, and improving the program. The program was fairly long (about 1400 lines of code), highly interactive, and had several different logical branches that it could take. Care had to be taken to protect the data base from involuntary improper input. (There is little hope of

protecting the data base from voluntary improper input if someone with that in mind gets as far as using the update program. In that case the idea is to prevent such people from getting access to the program in the first place by taking proper security precautions.) Since the eventual users of the program would be largely clerical types, there was need to make the program as "idiot proof" as possible. (This is not intended as a slur on people with clerical jobs. The fact is that Murphy's Law seems to hold rigidly with all computer systems. [Loosely stated, Murphy's Law says, "Whatever can go wrong, will."] Anyone who is not used to working with a computer can quickly find themselves in trouble through no fault of their own. The system designer must try to anticipate potential trouble spots as much as possible.)

During the time that the consultant worked on the update program, Dr. Driscoll was kept informed of the progress, sometimes on a daily basis. He was encouraged to try the various versions of the programs, and to think of as many possible variations as he could of the possible user input. In this way the users were kept actively involved in, and could contribute to, a large and integral part of a system which would eventually be their sole responsibility.

Some of the specific features of the update program

that were put in to aid the eventual users were as follows:

a. All user input is preceded by a request for the input, often with an explanation of the type of input expected. Then the prompt ==> is displayed to indicate that the user should begin entering data.

b. User input is entered one line at a time, and need not be left justified. In cases where user input must be of a specific type or value, the user is given three chances to make a correct entry, with successively stronger messages displayed after each incorrect entry. If proper input is not made after three chances, the program branches to its exit routine and stops executing.

c. When the user enters a member's name for the purpose of either changing or deleting the member's record, the computer finds the record, displays the name and address, and asks the user to verify that the correct member record has been found. This provides protection in case either the wrong name is input, or there are several members with the same name. In the latter case, the computer will look for another record with the same member name.

d. There are a limited number of geographic entities. Should the user enter a geographic location not listed in the data base (during an addition transaction), the computer will request that the location be

reentered, thus providing protection against a possible spelling mistake in the original entry.

e. As each transaction is completed, an entry is made in a data file which the user can check when all transactions have been completed for a particular run. There are actually three separate data files created, one each for additions, deletions, and modifications.

By contrast to the update program, the query program was simple to write, debug, and implement. As the update program was written first, some of the features of the update program could be adopted by the query program, and some of the trouble spots could be avoided. The main reasons the query program was so much easier, however, were that it is shorter, logically simpler, and accesses the data base in retrieval mode only, so that one does not have to worry about protecting the data base.

The query program uses no input files and creates no output files. All I/O is from a terminal. The user inputs the name of the member whose record is to be queried. The computer then acts in the same manner as described in part c above. Then the user is asked specifically about viewing every item in the member's record.

After testing the query program, Dr. Driscoll requested that in listing the rosters, groups, and

committees that a particular member belongs to, the reference number be included along with the description. Other than that, he accepted the program as written.

2.7.3 The Interface Program

As mentioned at the beginning of section 2.7, it would have been possible to write application programs which used the data base directly as their input, and which performed the batch processing required by the users. In fact, the consultant had started work in this area when the users decided not to go this way. The users were primarily motivated by ease of maintenance, and secondarily motivated by speed of the project. The batch application programs were already written to work with the old data file on the CDC computer. It would be relatively easy to adapt them to work on the DEC-20, and to use as input a file similar in format (but improved in content) to part of the old data file. It would also be faster and easier for the consultant to write an interface program to create such a file, than to write, debug, and test programs to perform the batch processing directly from the data base.

One can make a strong case for the method selected by the users in absolute terms. That is, it is quite possible that the users' decision was the correct one even

if processing efficiency was the sole criterion on which the decision was based. DBMS applications are designed to improve overall system efficiency, not specific programs. Application programs which use a DBMS tend to be I/O bound, hence slow in processing. In this case the users' requirements were for various types of sorted lists, and rarely involved the entire data base. The lists were generally for the members on selected rosters, groups, and/or committees. If application programs were written to use the data base directly as input, the same records would have to be accessed many times in various different sequences. To get a complete member record, several chains have to be traversed. The programs would have to check to make sure the member was on one of the selected rosters, groups, or committees. The data base design would have to be more complicated (see 2.6 part c).

The interface program allows the user to input the reference numbers of the rosters, groups, and/or committees to be used for a particular batch run. Then the program would make a temporary file of only that particular part of the data base needed. The batch application programs would then use this file as their input. As the creation of the temporary file did not require keeping track of as many things as would an application program that used the data base directly, the interface

program was easier to write and debug. From a users' point of view, the only requirement (other than having the necessary information) is to have the output in the proper format.

The flowchart and written documentation for the interface program are in appendix F. The user's manual is in appendix G. The program logic is actually fairly simple, and there was little trouble with either writing or debugging. The program gives the user the ability to have selected rosters, groups, and/or committees used in making a temporary file. It also allows the user three other options: (i) by entering ALL , the user will cause the program to use all rosters (hence all groups and committees also) in making the temporary file; (ii) by entering ABC , the user will cause the program to create a temporary file consisting of an alphabetical list of all the project members, along with their associated data; (iii) by entering ROS , the user will cause the program to create a temporary file consisting of a list of the rosters, groups, and committees similar to that in figure 2-1. This last provision was added later, at the request of the user, primarily to provide a vehicle to see how the internal reference numbers are matched with the appropriate rosters, groups, and committees.

2.8 Privacy and Security

Privacy and security did not play a major role in the development of this system. The information contained in the data base is neither financial nor particularly sensitive. Therefore the requirements for secure and private records are not as stringent as they would be had this not been the case.

2.8.1 Privacy

The operating system provides some protection from invasion of privacy by requiring a password to get at the project's computer directory. This feature is under the control of the computing center. Unauthorized programs cannot access the data base unless they contain the privacy keys of the subschema and the data base areas. These are under the control of the users, who may change them at will. Access to the authorized application programs is also under the control of the users.

2.8.2 Security

The consultant recommended that the users maintain a backup copy of the data base on tape, physically removed from the computer center. Therefore in the advent of physical loss of records, for whatever reason, the entire data base would not have to be recreated from

scratch. The users would have to request the computer center to make such a backup. The users have the responsibility of seeing to it that the backup copy is updated periodically.

Once the data base is up, only the update program actually modifies it. This program can be used only interactively, and in exclusive update mode (this update mode was the option selected by the users). Thus if the system should crash, at most one transaction, the current one, would be affected. Therefore there was no need to have the overhead of a transaction log to maintain data base integrity. Should a system crash occur, no further transactions would be allowed to take place. The utility software package DBMEND (see [7], and appendix H) can be used to restore the system to usable form, and the users can then take whatever steps are necessary to restore individual records. If they wish, they can use the utility software package DBINFO (see [7], and appendix H) to see precisely what is in the data base, including linkages.

CHAPTER 3

DBMS Application Guidelines

3.1 General Considerations

This chapter is aimed both at a consultant and a user in the environment described in section 1.3. It is written mainly to the consultant in order to provide a series of checkpoints in the development process. The user should also be aware of the checkpoints.

3.1.1 User Participation

The general organization of this chapter is to present the guidelines by category, where that is possible. There are, however, several points which cut across all the categories in the development process. The first and most important of these is: never work in a vacuum. Be sure the users are involved in every stage of the work, and are kept informed of the consultant's progress and problems on individual aspects of the work. While it is never a good idea to develop an information system without the user participating at least as an observer, in this type of situation it would be disastrous.

In a major systems development effort, with a large project team working, there is some hope of constructing a viable system with only minimal user involvement. A large group of professionals working together have the opportunity of "bouncing ideas around" until a good one surfaces. Potential trouble spots, solutions, user requirements, etc., can be anticipated, not by any one individual, but as a culmination of group discussions. The end result may not be precisely what the users wanted, but there is a reasonable possibility that it will be something they can live with.

In the situation considered here, there is no group. There are no team members to point out incorrect assumptions of user needs and desires, which are bound to occur. Only the users can say whether a proposed solution to a particular problem is viable. While there may be technical specialists the consultant can ask about specific details (as was the case in this case study), the users are the only ones with whom the consultant can discuss the problem as a whole. These discussions should be continual, as they will often uncover small but important details that were overlooked before. It has been this author's experience that, in general, the very fact of discussing a problem often seems to make the solution obvious. Finally, the consultant should not

lose contact with the users while involved with a specific task, e.g., writing a particular program, as a working relationship once lost may be difficult to regain.

There are positive reasons as well for keeping the users up to date on both the total picture and the individual tasks. Users can, and do, come up with valuable contributions to the development effort. While the users do not have technical DBMS expertise, they are the ones most familiar with their needs, and the old system. They know what worked well before and what did not.

3.1.2 Plan

Before embarking on the development effort, the consultant should have a general plan of attack. This should include a list of what has been completed, what is being worked on, and what is yet to be done. It should also include some peripheral items which may or may not be included, depending on how things go. The plan should not be construed as a rigid schedule, but as an aid in ordering priorities and in making sure important details are not overlooked. The very act of writing down the plan can often serve as a reminder of things forgotten. It can also bring conflicts and inconsistencies into focus.

3.1.3 Trade-offs

While working on the project, the consultant should constantly be on the lookout for possible compromises and trade-offs. The trade-offs should be user oriented. If it is possible, the users should be the ones who actually make the choice, after the consultant has fully gone over the possibilities. The expedient choice for the consultant will sometimes make the system more difficult to use for the users. On the other hand, very few things work out exactly as planned. The consultant should be flexible enough in his (or her) approach to be able to take the "second best" alternative, possibly at a later date. The best local solution is sometimes not the best global solution.

3.1.4 Odds and Ends

The consultant may find it helpful to keep careful notes about what has been done. The users undoubtedly have budgets to make and justify, and could easily require periodic formal reports and projections. The users, especially if the consultant has worked closely with them, may help with the reports.

Two final warnings to the consultant: (i) never be too sure you completely understand what the user wants; and (ii) be on the lookout for instances of Murphy's Law.

3.2 Pre-Design Phase

The pre-design phase of the development effort will be relatively short, but can set the tone for the remainder of the work. Misunderstandings at this point can come back to haunt the consultant later. The pre-design phase essentially means the initial contact, the job description, and the delineation of user and consultant responsibilities.

In the initial contact, the users will generally describe their particular needs. The consultant will give a general description about the capabilities of a DBMS. The consultant's description should be specific as to the types of activities a DBMS application is best suited for, but should avoid jargon and should not be too technical. It might be desirable to prepare a simple example to illustrate key points. The consultant should not be guilty of overselling (if for no other reason, than to avoid looking like an ass later). In fact, the consultant should point out potential alternative designs, along with their pros and cons. Obviously all of the above will probably not occur in a single meeting. The "initial contact" could be spread out over two or three meetings.

Once the decision to "go DBMS" has been made, the consultant must ascertain the users' willingness to work

with him (or her). If that is not forthcoming, the consultant should suddenly recall pressing concerns which make it impossible to undertake (a very apt word) a vast project with only half-vast methods of attack. Assuming that the users are willing to work with the consultant, the direct line of authority should be spelled out. It should be established from the first which individual(s) the consultant will be working directly with.

In conjunction with the principal user liason (hereafter referred to as the user), the job description and measures of effectiveness should be spelled out in some detail, preferably in writing. The functional specifications of the proposed system should be worked out, and prioritized in case it turns out that not all can be implemented. Possible future developments should also be discussed and prioritized, as they may affect the design and application programs.

In order to appraise the user of general progress, the consultant should set a general schedule and order of activities (not necessarily with expected times for each activity). Tasks which require the cooperation of outsiders, e.g., the computer center or special consultants on certain technical problems, should be identified as soon as possible. Arrangements for the cooperation should begin as soon as possible so that unnecessary delays will not occur later on.

In some cases it will actually be the users' responsibility to arrange for the outside assistance. The consultant's role in those cases should be a supportive one, perhaps with suggestions on how to best obtain the results. The consultant definitely should not abrogate any connection with the activity just because he (or she) is not directly involved. (The author's failure to make any suggestions on how to best obtain computer center cooperation on the tape conversion problem of the case study [see section 2.4] may have delayed implementation.)

3.3 Design

The design phase marks the beginning of the technical work on the DBMS application. It is a phase that never really ends, even after implementation. Therefore the most important guideline for the consultant is to keep the design flexible, so that changes can be accommodated.

3.3.1 A Workable Design

Producing a workable design will undoubtedly be an iterative back and forth process between the consultant and the user. Some of the key factors affecting the design, in pretty much their order of importance, are the

functional specifications of the system, processing efficiency for the application programs, future application developments, and the size of the data base, which includes the space for records, overhead, and future expansion. Also, there probably is some overriding fundamental problem which caused the users to consider a DBMS application in the first place. (In the case study, it was the update problem.) Any design must first and foremost attack that problem.

Each design iteration should consist of the consultant presenting a possible design to the user, indicating what each of the data items are, showing the linkages between record types, and giving a general idea of how the functional specifications will be met. The consultant should point out possible trade-offs and compromises. A clear schematic diagram (see [11]) will aid the user in deciding which data items should be left out or added, which important factors were somehow overlooked in earlier discussions, which linkages were unimportant and should never have been made, and, in general, clearing up earlier misunderstandings.

3.3.2 The Old System

If the DBMS application is replacing an earlier computer system, the old system can be a valuable guide

toward deciding which data items go with which record types, how linkages should be made, and even such mundane matters as naming of record and set types. By making the new system bear a superficial resemblance to the old, even at a slight loss in overall efficiency, the consultant will aid the users in understanding the system. This can possibly mean the difference between a workable going concern, and an elegant disaster.

3.3.3 The Loading Plan

During the design phase, the consultant should begin firming a plan for loading the data base. This requires the close cooperation of the user, as the user should be responsible for providing the data for the loading. The loading plan and the data to be loaded can affect both the system design and the logic of the application program used for the loading.

Making the user responsible for providing the loading data is important for several reasons. Among them are:

(i) The user is the only one qualified to know what data is valid, and what data is not valid. Having to provide the loading data will force the user to clearly think through what should be included, and what should not.

(ii) If the consultant has to assemble the loading

data, he (or she) will not be able to work on other tasks for which no one else is qualified, thus delaying the entire project.

(iii) Gathering the loading data will aid the user in understanding the design and usage of the new system.

(iv) Gathering the data will aid the user in thinking about needed application programs.

(v) Finally, gathering the data will make the user aware (if he was not already) of the computer system axiom "garbage in, garbage out."

3.4 Application Programs

The application programs include the loading program, probably to be used only once, and the programs written to satisfy the functions of the system, which will probably be used many times. The loading program is a special case, as it is the only one where the consultant is not completely in control. The logic of the loading program depends on the data provided by the users. For testing and debugging the loading program, the consultant must know the format and general contents of the loading data. The user can be a big help here in providing the details necessary, and possibly providing the test data itself (as was the case in the case study).

If the user does provide the test data for the loading program, the consultant must make sure that all of the logical branches of the program are tested.

It is probably better not to use the real data as test data because the sheer volume of the real data precludes complete checking of linkages, data item values, etc. The test data should be small enough to test and check details, and large enough to cover most situations. As there will be some kind of an update program, the test data for the loading program does not have to cover all situations for all the application programs. An update program can load other remaining test data.

The importance of proper test data and procedures cannot be overemphasized. The consultant will go away and leave the users "holding the bag" after system implementation. Even if the system works perfectly, there will be problems if the user manuals are not very good. Thorough testing with good test data is very helpful in writing good user manuals. More will be said about the manuals in the next section.

The application programs themselves, especially the interactive ones, should be as complete and "idiot proof" as possible. User input should be simple and described fully. The machine should do as much of the

work as possible. There is a good chance that the people the programs will be interacting with will not be technically oriented. They will not be readily able to follow involved logical sequences. The programs should anticipate user input as much as possible, and guide that input with appropriately displayed messages. It is probably a good idea to sacrifice I/O efficiency, and have the terminal users input only one line at a time. The user liason should test the interactive programs himself (or herself). If that works out, it would be desirable to get others to try them also, if possible some of the people who will be using them after implementation.

After the user has checked out the application programs, it is probable that he (or she) will have suggestions to make. At this point the consultant will have to be very careful. The user probably would be referring to one application program. If the suggestions merit inclusion into the system, and involve changes only in the particular program, there will be no real problem. If the suggestions require changes in the schema, however, there may be a big problem. Changes in the schema, especially those that would change linkages and/or methods of access, can affect all the application programs. The ramifications of a single change can be far reaching in terms of the entire system, and could cause substantial delays in return for a minor gain. This is not to say

that the changes should not be made, only that they should be investigated fully before they are attempted. Then the alternatives should be presented to the user, who should make the decision as to whether or not the changes should be implemented. (The user may claim that the suggested changes should have been incorporated into the system in the first place. That may be true, but it is beside the point. The issue at this time would be whether it is better to undo work already done, or live with what exits.)

Batch processing applications using a DBMS data base require some planning. DBMS application programs tend to be I/O bound. When properly used, a DBMS application should increase overall system efficiency, but may decrease processing speed on some individual programs. One possible solution is the one taken in this case study: to create a temporary file, and then use that as the input for the batch runs. Another possibility is simply to schedule batch runs for times when the computer is not busy. The decision may be just to live with the situation, and not do anything special. The important point is to recognize the problem, to make sure the user recognizes it also, and to make an active decision rather than to just fall into one by default.

3.5 Documentation and Manuals

3.5.1 Documentation

The most important thing that can be said about documentation is: DO IT! Check with the user to see if there are any documentation standards to be met. If there are, fine. If not the consultant will have to decide on his (or her) own standards. In addition to the usual source listings, the documentation should include at least a schema diagram, a data dictionary, flow charts of the application programs, and a written narrative explaining the logical branches in each program.

Documentation samples should be shown to the user to see if he (or she) can follow the logic. As the user will be responsible for maintaining the system after implementation, the more complete the documentation, the better. In the case study, in order to make cross referencing easier, the author alphabetized the various procedures in the narrative, and indicated by line number where they could be found on the source listing.

3.5.2 General Information Manual

As mentioned several times earlier, the users are not familiar with DBMS's. They will have to gain some expertise in the area in order to maintain the system.

One way the consultant can help is to provide a general information manual. The purpose of this manual is to aid the users in using the DBMS and its facilities by aiding them in reading the vendor manuals. (For some reason vendors seem to pride themselves on providing difficult to read manuals.) The application programs will probably be written in a host language (COBOL, FORTRAN, PL/1, etc.) using special data base commands. The general information manual should explain the logic involved in using the most common of the commands, e.g., common sequences in which they are used. The manual should also explain how to use the recovery techniques and the information gathering techniques of the DBMS. There will inevitably be system crashes. There will also be times when the user wants to check on such details as data item values, linkages, available space, and other items of interest about the data base. (If the overall computer system allows it, it might be a good idea for the consultant to leave the test data base as a separate entity from the real one. That way the users will have something to practice on which resembles the real thing, and which will cause no real harm if a partially debugged program mangles the data base.)

3.5.3 Application Program Manuals

Good user manuals for the application programs, especially the interactive programs, are critical to the proper running of the system. A perfect system is of no value if it cannot be used.

A user manual for an application program should explain the general purpose of the program, the different types of output possible, and how, when, and in what form user input should be. If the program is interactive, the manual should explain each terminal message from the machine (in some detail). A terminal message which says:

ENTER THE ROSTER TYPE

may have been crystal clear in meaning to both the consultant and the user liason when the manual was written, but could easily be gibberish to the terminal user.

The manuals should make clear what user input is valid in each case. To a certain extent, the consequences of each valid input should be explained. To a larger extent, the consequences of invalid input should be spelled out. The manuals should explain what the user can do in order to recover from invalid input, and what to do if the system gets tempermental and refuses to take valid input (as sometimes happens in the best of systems).

3.6 Implementation

At this point the design has been finalized, the application programs have been written, tested, and debugged, documentation and manuals have been completed, and the system is all set to go, right? If you buy that, perhaps you would also be interested in these gold mine stocks.... In the development effort to this point, there have undoubtedly been a number of minor changes whose cumulative effect might have thrown off some of the initial calculations. One of the first tasks is to recheck the size of the data base, by area if the DBMS breaks up the data base into areas. Make sure the input formats of the loading data agrees with the format specified in the loading program.

After the loading program has been run, check the contents of the data base for proper data item values, linkages, amount of free space, etc. Then check the application programs. Programs which work well on a small data base might not do so on a large one.

Finally, before the system is turned over, the consultant should go over the components of the system with the users. Final user questions should be answered. And lastly, the consultant should make sure the users have understood all the documentation and manuals, including the vendor manuals.

3.7 Items for Further Study

The knowledgeable reader will have noticed that there are several key items either not mentioned in the previous sections of this chapter, or mentioned superficially only. Among these are such things as simultaneous updating of the data base, transaction logging, security procedures, and extensive system testing. These items, and others, were left up to now because the case study either did not require them, or did not provide enough insight from which to generalize. Such items will be discussed briefly in this section, sometimes with outside references mentioned. All the items mentioned in this section require further study. Also, since one case study can be misleading, the items in the previous sections merit further study.

The order of presentation here is (more or less) the order in which a consultant would have to consider the items in a development effort.

a. The assumption of this thesis is that the consultant would be working largely by himself on the technical details of the development. These guidelines, or a modified version of them, might be valid for a small (two or three people) team, which might be preferable for a system slightly larger than the one envisioned here.

b. The case study was particularly well suited

for a single data base in a DBMS application (see section 2.4). Other systems might require more extensive investigation into alternative file designs, multiple data bases, or a DBMS database used in conjunction with another file, such as an index file.

c. For the reasons mentioned in section 2.8, transaction logging of update runs was not included in the case study. Any system having batch or simultaneous on-line updating would have to consider transaction logging.

d. Should transaction logging be used, then recovery procedures much more sophisticated than the ones used in this case study would have to be implemented.

e. If the situation is such that simultaneous updating will be done on-line, i.e., two or more users will be updating the data base from different terminals at the same time, then careful programming is called for. Yourdon in [15] discusses this problem. The DBMS software may take care of many of the problems that Yourdon alludes to, but the situation still requires more careful planning than the case study did, and will alter the sturcture of the application programs.

f. If updating is done in batch mode, then backup and recovery, and update program input, will be different enough from that in the case study to require

separate study.

g. If there is extensive batch programming of any type, efficiency considerations with respect to the design of the schema and the application programs have to be taken into account much more than they were in the case study.

h. If there are organization documentation standards, the documentation suggested here should be compatible, but it might be desirable to look into that some more. The documentation suggested here might need some modification.

i. If the data contained in the data base is particularly sensitive and/or financial in nature, then extensive security and privacy measures are called for. These will tend to be hardware and/or software oriented, and depend on what is available at a given computer installation. There may, however, be some general guidelines. Martin in [13], among others, has written an entire book on the subject.

j. If the system is complicated and/or is apt to have multiple users at the same time, system flowcharts can help to plan the system. Implementation can take place in carefully planned stages. Finally, extensive system testing is called for before the system is turned over to the users.

3.8 Summary

In the previous sections, guidelines were presented in narrative style. That approach was taken so that the guidelines could be justified as much as possible, and so that repercussions of particular actions could be discussed. This section just lists the key tasks in more or less their proper order.

A. General

i. Involve the user in all stages.

ii. Keep an updated general plan.

iii. Watch for compromises and trade-offs.

iv. Always try to ease things for the users

v. Remember: If something can go wrong, it will; if nothing can go wrong, something will anyway.

B. Pre-design

i. Get the general idea of the users' problems.

ii. Describe DBMS capabilities to the users.

iii. Establish which individual will be the principal user contact, and develop a close working relationship.

iv. Get functional specifications detailed and prioritized.

C. Design

i. Work out the schema in conjunction with the user.

ii. Begin firming the loading plan.

iii. Investigate whether transaction logging is necessary.

D. Application Programs

i. Work out the loading plan and program with the user.

ii. Plan for test data and program testing.

iii. Write, test, and debug programs.

iv. Have the user test the interactive programs.

v. Consider the problem of batch processing efficiency.

vi. Consider the problem of simultaneous update.

vii. Consider the backup and recovery problem.

E. Documentation and Manuals

i. Check documentation standards.

ii. Write data dictionary.

iii. Write flow charts and documentation narratives.

iv. Consider the need for system flow charts.

v. Write general information manual.

vi. Write application program manuals.

F. Implementation

i. Plan the implementation schedule,

ii. Check the size of the data base.

iii. Check the format of the data for the loading program.

iv. Load the data base.

v. Check the loaded data base,

vi. Perform necessary system testing,

vii. Review security and privacy features.

viii. Review the system with the users,

ix. Turn the system over to the users.

Bibliography

[1] Beedle, L. S. A Time to Build Up..., A pamphlet published by the Council on Tall Buildings and Urban Habitat, Bethlehem, Pennsylvania, August, 1977.

[2] _____, RSVP, An introduction to the Tall Buildings Project for new staff members, Bethlehem, Pennsylvania, May, 1975.

[3] Brinker, T. W. "Controlling a Large Interdisciplinary International Research Project" Master's Thesis, Lehigh University, 1976.

[4] Burch, J. G. and Strater, F. R. Information Systems: Theory and Practice, Hamilton Publishing Company, Santa Barbara, California, 1974.

[5] Cincom Systems, Inc., TOTAL/7 Reference Manual, Cincinnati, Ohio, 1976.

[6] Datapro Research Corporation, A Buyer's Guide to Data Base Management Systems, Delran, New Jersey, 1975.

[7] DECSYSTEM, DATA BASE MANAGEMENT SYSTEM Administrators Procedures Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1977.

[8] _____, DATA BASE MANAGEMENT SYSTEM Programmer's Procedures Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1977.

[9] IBM, IMS/VS Version 1, General Information Manual, seventh edition, White Plains, New York, 1977.

[10] Kroenke, P. Database Processing, Fundamentals, Modeling, Applications, Science Research Associates, Chicago, Palo Alto, Toronto, Henley-on-Thames, Sydney, Paris, Stuttgart, 1977.

[11] Martin, J. Computer Data-Base Organization, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[12] _____. Principles of Data-Base Management, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[13] _____. Security, Accuracy, and Privacy in Computer Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

[14] National Bureau of Standards Handbook 113, CODASYL Data Description Language Journal of Development, U. S. Department of Commerce, Washington, D. C., 1974.

[15] Yourdon, E. Design of On-Line Computer Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

APPENDIX A

SCHEMA DATA DESCRIPTION

```
IMAGES BY COMMAND.
NOTE UNANTICIPATED.
INTERCEPT BIND.
ASSIGN TALL-AREA TO LISTS
RPP 200
BUFFER 4
CALC 2 RPP
FIRST PAGE IS 800
LAST PAGE IS 2200
PAGE SIZE IS 512 WORDS.
ASSIGN NAIM-AREA TO NAIMS
RPP 100
BUFFER 4
CALC 2 RPP
FIRST PAGE IS 100
LAST PAGE IS 600
PAGE SIZE IS 512 WORDS.

SCHEMA NAME IS TALL-B.

AREA NAME IS TALL-AREA
        PRIVACY LOCK EXCLUSIVE UPDATE IS FIXEM
        PRIVACY LOCK FOR RETRIEVAL IS READEM.
AREA NAME IS NAIM-AREA
        PRIVACY LOCK EXCLUSIVE UPDATE IS FIXEM
        PRIVACY LOCK FOR RETRIEVAL IS READEM.
```

```
RECORD NAME IS ROSTER-REC
        LOCATION MODE IS CALC USING ROS-NUM
        DUPLICATES NOT ALLOWED
        WITHIN TALL-AREA.
                02  ROS-NUM                    PIC 999.
                02  R-TYPE                     PIC XX.
                02  ROSTER-IDENT               PIC X(6).
                02  R-DETAIL-1                 PIC X(32).
                02  R-DETAIL-2                 PIC X(30).
                02  ROS-OUTPUT                 PIC X(4).
RECORD NAME IS R-COMMITTEE
        LOCATION MODE IS CALC USING R-COMM-NUM
        DUPLICATES NOT ALLOWED
        WITHIN TALL-AREA.
                02  R-COMM-NUM                 PIC 999.
                02  R-COMM-TYPE                PIC XX.
                02  R-COMM-IDENT               PIC X(6).
                02  R-COMM-DETAIL-1            PIC X(32).
                02  R-COMM-DETAIL-2            PIC X(30).
                02  R-COMM-OUTPUT              PIC XXXX.
RECORD NAME IS GROUP-REC
        LOCATION MODE IS CALC USING G-NUM
        DUPLICATES NOT ALLOWED
        WITHIN TALL-AREA.
                02  G-NUM                      PIC 999.
```

```
                02  G-TYPE                          PIC XX.
                02  GROUP-IDENT                     PIC X(6).
                02  GROUP-DETAIL-1                  PIC X(32).
                02  GROUP-DETAIL-2                  PIC X(30).
                02  G-OUTPUT                        PIC XXXX.
RECORD NAME IS GROUP-COMMITTEE
        LOCATION MODE IS CALC USING G-COMM-NUM
        DUPLICATES NOT ALLOWED
        WITHIN TALL-AREA.
                02  G-COMM-NUM                      PIC 999.
                02  G-COMM-TYPE                     PIC XX.
                02  G-COMM-IDENT                    PIC X(6).
                02  G-COMM-DETAIL-1                 PIC X(32).
                02  G-COMM-DETAIL-2                 PIC X(30).
                02  G-COMM-OUTPUT                   PIC XXXX.
RECORD NAME IS NAME-GROUP-REC
        LOCATION MODE IS VIA GROUP-NAME-SET
        WITHIN TALL-AREA.
                02  GROUP-RESPONS                   PIC X(6).
                02  GROUP-NAME                      PIC X(47).
RECORD NAME IS NAME-G-COMM-REC
        LOCATION MODE IS VIA G-COMM-NAME-SET
        WITHIN TALL-AREA.
                02  G-COMM-RESPONS                  PIC X(6).
                02  G-COMM-NAME                     PIC X(47).
```

```
RECORD NAME IS NAME-R-COMM-REC
        LOCATION MODE IS VIA R-COMM-NAME-SET
        WITHIN TALL-AREA.
                02  R-COMM-RESPONS              PIC X(6).
                02  R-COMM-NAME                 PIC X(47).
RECORD NAME IS NAME-ROSTER-REC
        LOCATION MODE IS VIA ROSTER-NAME-SET
        WITHIN TALL-AREA.
                02  ROSTER-RESPONS              PIC X(6).
                02  ROSTER-NAME                 PIC X(47).
RECORD NAME IS STATE-COUNTRY-REC
        LOCATION MODE IS CALC USING STATE-COUNTRY
        DUPLICATES NOT ALLOWED
        WITHIN TALL-AREA.
                02  STATE-COUNTRY SIZE 32 USAGE DISPLAY-6.
RECORD NAME IS CITY-REC
        LOCATION MODE IS VIA CITY-COUNTRY-SET
        WITHIN TALL-AREA.
                02  CITY                        PIC X(32).
RECORD NAME IS ORGANIZATION-REC
        LOCATION MODE IS CALC USING ORGANIZATION
        DUPLICATES ALLOWED
        WITHIN TALL-AREA.
                02  ORGANIZATION                PIC X(34).
```

```
RECORD NAME IS NAME-FILE
        LOCATION MODE IS CALC USING NAIM
        WITHIN NAIM-AREA.
                02  NAIM SIZE 47 USAGE DISPLAY-6.
                02  TITLE                          PIC X(17).
                02  SECOND-LINE                    PIC X(32).
                02  THIRD-LINE                     PIC X(32).
                02  FOURTH-LINE SIZE 33 USAGE DISPLAY-6.
                02  PHONE                          PIC X(10).
                02  START-DATE                     PIC X(4).
                02  END-DATE                       PIC X(4).
SET NAME IS ROSTER-GROUP-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS ROSTER-REC
        MEMBER IS GROUP-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS ROSTER-COMM-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS ROSTER-REC
        MEMBER IS R-COMMITTEE
                MANDATORY AUTOMATIC
```

```
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS GROUP-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
                OWNER IS GROUP-REC
        MEMBER IS NAME-GROUP-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS GROUP-NAME
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS GROUP-COMM-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS GROUP-REC
        MEMBER IS GROUP-COMMITTEE
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS G-COMM-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS GROUP-COMMITTEE
        MEMBER IS NAME-G-COMM-REC
```

```
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS G-COMM-NAME
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS NAME-G-COMM-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS NAME-FILE
        MEMBER IS NAME-G-COMM-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS NAME-GROUP-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS NAME-FILE
        MEMBER IS NAME-GROUP-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS R-COMM-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS R-COMMITTEE
```

```
        MEMBER IS NAME-R-COMM-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS R-COMM-NAME
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS NAME-R-COMM-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS NAME-FILE
        MEMBER IS NAME-R-COMM-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS ROSTER-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS ROSTER-REC
        MEMBER IS NAME-ROSTER-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS ROSTER-NAME
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS NAME-ROSTER-SET
```

```
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS ALWAYS NEXT
        OWNER IS NAME-FILE
        MEMBER IS NAME-ROSTER-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
        SELECTION CURRENT.
SET NAME IS CITY-COUNTRY-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS STATE-COUNTRY-REC
        MEMBER IS CITY-REC
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS CITY
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS CITY-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS CITY-REC
        MEMBER IS NAME-FILE
                OPTIONAL MANUAL
                LINKED TO OWNER
                ASCENDING KEY IS NAIM
```

```
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS ORGAN-NAME-SET
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED WITHIN RECORD-NAME
        OWNER IS ORGANIZATION-REC
        MEMBER IS NAME-FILE
                OPTIONAL MANUAL
                LINKED TO OWNER
                ASCENDING KEY IS NAIM
                DUPLICATES ALLOWED
        SELECTION CURRENT.
SET NAME IS ALPHABETICAL
        MODE IS CHAIN LINKED TO PRIOR
        ORDER IS SORTED
        DUPLICATES ALLOWED
        OWNER IS SYSTEM
        MEMBER IS NAME-FILE
                MANDATORY AUTOMATIC
                ASCENDING KEY IS NAIM.
SUB-SCHEMA NAME IS UNIVERSAL
        PRIVACY LOCK IS ALLAR.
AREA SECTION.
        COPY ALL AREAS.
```

```
RECORD SECTION.
        01  ROSTER-REC.
        01  R-COMMITTEE.
        01  GROUP-REC.
        01  GROUP-COMMITTEE.
        01  NAME-GROUP-REC.
        01  NAME-G-COMM-REC.
        01  NAME-R-COMM-REC.
        01  NAME-ROSTER-REC.
        01  STATE-COUNTRY-REC.
              02 STATE-COUNTRY.
                    03 COUNTRY                  PIC X(5).
                    03 STATE                    PIC X(27).
        01  CITY-REC.
        01  ORGANIZATION-REC.
        01  NAME-FILE.
              02 NAIM.
                    03 LAST-NAME                PIC X(20).
                    03 FIRST-NAME.
                          04 F-INITIAL          PIC X.
                          04 REST-FIRST         PIC X(13).
                    03 MIDDLE.
                          04 M-INITIAL          PIC X.
                          04 REST-MIDDLE        PIC X(12).
```

```
            02 TITLE.
            02 SECOND-LINE.
            02 THIRD-LINE.
            02 FOURTH-LINE.
                 03 CHAR-28                    PIC X(28).
                 03 ZIP                        PIC X(5).
            COPY OTHERS.

SET SECTION.
        COPY ALL SETS.

END-SCHEMA
```

APPENDIX B

DATA DICTIONARY

DATA DICTIONARY OF TERMS AND DATA NAMES USED IN THE TALL BUILDING SCHEMA

ALLAR: The privacy lock for the part of the data base called the UNIVERSAL sub-schema. This lock may be changed at the option of the user, but has to be changed in both the schema and the application programs.

ALPHABETICAL: The system name given the set of member records, i.e., the collection of NAME-FILE records. The set is ordered alphabetically by member name.

AREA: A sub-collection of the entire data base. It is usually isolated physically in order to expedite expected processing.

CHAR-28: The first 28 characters of the 33 character data item FOURTH-LINE. FOURTH-LINE is an item of the record type NAME-FILE.

CITY: The 32 character alphanumeric data item in a CITY-REC record containing the name of the city.

CITY-COUNTRY-SET: The name of the collection of cities and countries or states. A particular occurrence would contain a STATE-COUNTRY-REC record together with the CITY-REC records of the cities in the state or country given in the occurrence of the STATE-COUNTRY-REC record.

CITY-NAME-SET: The name of the collection of cities and associated memgers of the project. A particular occurrence would contain a CITY-REC record and the NAME-FILE records of the members located in the particular city.

CITY-REC: The name of the record type which has the records of the cities of the members. Its only data item is CITY.

COUNTRY: The data item name given the first five characters of the data item STATE-COUNTRY. Should the occurrence of the associated STATE-COUNTRY-REC record be a state in the United States, COUNTRY has the value ZZUSA . Otherwise its value is just the first five characters of the country given in STATE-COUNTRY.

END-DATE: The four character alphanumeric data item in a NAME-FILE record giving the date that the member ended his association with the project.

The date is in the form YYMM.

EXCLUSIVE UPDATE: The mode used for application programs which can modify the data base. When used, no other application program can access the data base for any purpose.

F-INITIAL: The first initial of the first name of a member. It is the first character of the data item FIRST-NAME, which is part of the data item NAIM, which is the item in a NAME-FILE record containing the name of the member.

FIRST-NAME: The 14 character alphanumeric item containing the first name of a member. It is part of the item NAIM, which is part of the record type NAME-FILE.

FIXEM: The privacy lock for application programs using exclusive update mode. This may be changed at the option of the user, but should be changed in both the schema description and the appropriate application programs.

FOURTH-LINE: The 33 character alphanumeric data item in a NAME-FILE record containing the fourth line of the member's four line address.

G-COMM-DETAIL-1: The 32 character data item in a GROUP-

COMMITTEE record giving the first part of the description of the group committee.

G-COMM-DETAIL-2: The 30 character data item in a GROUP-COMMITTEE record giving the second part of the description of the group committee.

G-COMM-IDENT: The data item within a GROUP-COMMITTEE record giving the last six characters of the eight character identifier used under the old system to identify the group committee.

G-COMM-NAME: The 47 character alphanumeric data item in a NAME-G-COMM-REC record giving the name of a member on the associated group committee.

G-COMM-NAME-SET: The name of the collection of group committees and project members. A particular occurrence of the set consists of a GROUP-COMMITTEE record together with the NAME-G-COMM-REC records of the members on the particular group committee.

G-COMM-NUM: The three character numeric data item in a GROUP-COMMITTEE record giving the number used for internal location purposes of the GROUP-COMMITTEE record.

C-COMM-OUTPUT: The four character data item within a GROUP-COMMITTEE record giving a user defined code used in applications to identify the particular group committee.

G-COMM-RESPONS: The six character alphanumeric data item

in a NAME-G-COMM-REC record identifying the responsibility of the member in the associated group committee.

G-COMM-TYPE: The data item within a GROUP-COMMITTEE record giving the first two characters of the eight character identifier used under the old system to identify the group committee.

G-NUM: The 3 digit numeric data item within a GROUP-REC record giving the number used for internal location purposes of the GROUP-REC record.

G-OUTPUT: The four character alphanumeric data item within a GROUP-REC record giving a user defined code used in applications to identify the particular group.

G-TYPE: The data item within a GROUP-REC record giving the first two characters of the eight character identifier used under the old system to identify the group. It usually has the value GP.

GROUP-COMM-SET: The name of the collection of groups and group committees. A particular occurrence of the set consists of a GROUP-REC record and the GROUP-COMMITTEE records of the committees of the group.

GROUP-COMMITTEE: The name of the record type of a committee contained in a roster which is broken up into

groups, or in other words the record type of a group committee. A GROUP-COMMITTEE record has as data items G-COMM-NUM, G-COMM-TYPE, G-COMM-IDENT, G-COMM-DETAIL-1, G-COMM-DETAIL-2, and G-COMM-OUTPUT.

GROUP-DETAIL-1: The 32 character alphanumeric data item within a GROUP-REC record used to give the first part of the description of the group.

GROUP-DETAIL-2: The 30 character alphanumeric data item within a GROUP-REC record used to give the second part of the description of the group.

GROUP-IDENT: The data item within a GROUP-REC record giving the last six characters of the eight character identifier used under the old system to identify the group.

GROUP-NAME: The 47 character data item within a NAME-GROUP-REC record giving the name of a member in the associated group.

GROUP-NAME-SET: The name of the collection of groups and project members. A particular occurrence of the set consists of a GROUP-REC record and the NAME-GROUP-REC records of various members in the group.

GROUP-REC: The name of the record type of a group. GROUP-REC has as data items G-NUM, G-TYPE, GROUP-IDENT, GROUP-DETAIL-1, GROUP-DETAIL-2, and G-OUTPUT.

GROUP-RESPONS: The six character alphanumeric data item within a NAME-GROUP-REC record giving the responsibility of the member in the associated group.

LAST-NAME: The 20 character item giving the last name of a project member. LAST-NAME is a sub-field of the data item NAIM, which is an item of the record type NAME-FILE.

M-INITIAL: The first initial of the middle name of a project member. M-INITIAL gives the first character of the data item MIDDLE. MIDDLE is a sub-field of the data item NAIM, which is an item of the record type NAME-FILE.

MIDDLE: The 13 character item giving the middle name of a project member. MIDDLE is a sub-field of the data item NAIM, which is an item of the record type NAME-FILE.

NAIM: The 47 character alphanumeric item in a NAME-FILE record giving the full name of a project member. NAIM is sub-divided into the fields LAST-NAME, FIRST-NAME, and MIDDLE.

NAIM-AREA: The area of the data base consisting of all the NAME-FILE records.

NAME-FILE: The record type of the individual project members. NAME-FILE has as data items NAIM, TITLE, SECOND-LINE, THIRD-LINE, FOURTH-LINE, PHONE, START-DATE, and END-DATE.

NAME-G-COMM-REC: The record type linking a member with a group committee. It has as fields G-COMM-RESPONS and G-COMM-NAME.

NAME-G-COMM-SET: The name of the collection of member records and links to associated group committees. A particular occurrence of the set consists of a NAME-FILE record together with a NAME-G-COMM-REC record for each group committee that the member is on.

NAME-GROUP-REC: The record type linking a member with a group. It has fields GROUP-RESPONS and GROUP-NAME.

NAME-GROUP-SET: The name of the collection of member records and links to associated groups. A particular occurrence of the set consists of a NAME-FILE record together with a NAME-GROUP-REC record for each group the member is specifically associated with.

NAME-R-COMM-REC: The record type linking a member with a

roster committee. It has as fields R-COMM-RESPONS and R-COMM-NAME.

NAME-R-COMM-SET: The name of the collection of member records and links to associated roster committees. A particular occurrence of the set consists of a NAME-FILE record together with a NAME-R-COMM-REC record for each roster committee that the member is on.

NAME-ROSTER-REC: The record type linking a member with a roster. It has as fields ROSTER-RESPONS and ROSTER-NAME.

NAME-ROSTER-SET: The name of the collection of member records and links to associated rosters. A particular occurrence of the set consists of a NAME-FILE record together with a NAME-ROSTER-REC record for each roster the member is specifically associated with.

ORGAN-NAME-SET: The name of the collection of organizations and associated member records. A particular occurrence of the set consists of an ORGANIZATION-REC record together with a NAME-FILE record for each project member in the organization.

ORGANIZATION: The 34 character alphanumeric data item in

an ORGANIZATION-REC record giving the name of the organization.

ORGANIZATION-REC: The record type of organization records for the project members' organizations. Its only date item is ORGANIZATION. Organizations which are also project members would have a NAME-FILE record in addition to an ORGANIZATION-REC record.

PHONE: The 10 character alphanumeric data item in a NAME-FILE record giving the member's phone number.

R-COMM-DETAIL-1: The 32 character alphanumeric data item in an R-COMMITTEE record giving the first part of the description of the roster committee.

R-COMM-DETAIL-2: The 30 character alphanumeric date item in an R-COMMITTEE record giving the second part of the description of the roster committee.

R-COMM-IDENT: The data item within an R-COMMITTEE record consisting of the last six characters of the eight character identifier used under the old system to identify the roster committee.

R-COMM-NAME: The 47 character alphanumeric data item in a NAME-R-COMM-REC record giving the name of a member associated with the particular roster committee.

R-COMM-NAME-SET: The name of the collection of roster committees and associated project members. A particular occurrence of the set consists of an R-COMMITTEE record together with the NAME-R-COMM-REC records of the members on the committee.

R-COMM-NUM: The 3 digit numeric data item in an R-COMMITTEE record giving the number used for internal location purposes of the R-COMMITTEE record.

R-COMM-OUTPUT: The 4 character alphanumeric data item within an R-COMMITTEE record giving the user defined code used in identifying the roster committee in applications.

R-COMM-RESPONS: The six character alphanumeric data item within a NAME-R-COMM-REC record giving the responsibility of the member on the particular roster committee.

R-COMM-TYPE: The data item within an R-COMMITTEE record giving the first two characters of the eight character identifier used under the old system to identify a roster committee. It usually identifies the type of committee.

R-COMMITTEE: The name of the record type used for the record of a committee contained in a roster which

is not broken up into groups. Its fields are R-COMM-NUM, R-COMM-TYPE, R-COMM-IDENT, R-COMM-DETAIL-1, R-COMM-DETAIL-2, and R-COMM-OUTPUT.

R-DETAIL-1: The 32 character alphanumeric data item within a ROSTER-REC record giving the first part of the description of the roster.

R-DETAIL-2: The 30 character alphanumeric data item within a ROSTER-REC record giving the second part of the description of the roster.

R-TYPE: The data item within a ROSTER-REC record giving the first two characters of the eight character identifier used under the old system to identify the roster. It usually describes the type of roster.

READEM: The privacy lock for application programs using retrieval mode. It may be changed at the option of the user, but should be changed in both the schema description and the appropriate application programs.

REST-FIRST: The last 13 characters of the 14 character data item FIRST-NAME. FIRST-NAME is part of the data item NAIM, which is a field of the record type NAME-FILE.

REST-MIDDLE: The last 12 characters of the 13 character

data item MIDDLE. MIDDLE is part of the data item NAIM, which is a field of the record type NAME-FILE.

RETRIEVAL: The usage mode for application programs which access the data base without changing it. While in use, updates under the mode exclusive update are not permitted, but other users may use retrieval mode.

ROS-NUM: The 3 digit numeric data item in a ROSTER-REC record giving the number used for internal location purposes of the roster.

ROS-OUTPUT: The 4 character alphanumeric data item within a ROSTER-REC record giving the user defined code used in applications to identify the roster.

ROSTER-COMM-SET: The name of the collection of rosters and associated committees. A particular occurrence of the set consists of a ROSTER-REC record and R-COMMITTEE records for the committees of the roster.

ROSTER-GROUP-SET: The name of the collection of rosters and associated groups. A particular occurrence of the set consists of a ROSTER-REC record and GROUP-REC records for the groups contained in

the roster.

ROSTER-IDENT: The data item consisting of the last six characters of the eight character identifier used under the old system to identify the roster.

ROSTER-NAME: The 47 character alpanumeric data item within a NAME-ROSTER-REC record giving the name of a member on the particular roster.

ROSTER-NAME-SET: The name of the collection of roster and member names. A particular occurrence of the set consists of a ROSTER-REC record together with the NAME-ROSTER-REC records of the project members on the roster.

ROSTER-REC: The record type containing the records of the rosters. Its fields are ROS-NUM, R-TYPE, ROSTER-IDENT, R-DETAIL-1, R-DETAIL-2, and ROS-OUTPUT.

ROSTER-RESPONS: The six character alphanumeric data item within a NAME-ROSTER-REC record giving the responsibility of the member on the associated roster.

SCHEMA: The logical description of the data base, giving the records, data items, areas, sets, linkages, and sub-schemas.

SECOND-LINE: The 32 character alphanumeric data item

contained in a NAME-FILE record type. It gives the second line of the member's four line address.

START-DATE: The four character alphanumeric data item contained in a NAME-FILE record type. It gives the starting date of the member with the project in the form YYMM.

STATE: The last 27 characters of the data item STATE-COUNTRY. If STATE-COUNTRY-REC gives the record of a state in the United States, STATE has the name of the state. Otherwise, it is just a continuation of the country name.

STATE-COUNTRY: The 32 character alphanumeric data item in STATE-COUNTRY-REC giving the name of the state or country. It is broken up into COUNTRY and STATE. If COUNTRY has the value ZZUSA, then STATE has the name of a state in the United States. Otherwise, STATE-COUNTRY has the name of a country other than the United States.

STATE-COUNTRY-REC: The name of the record type of states and countries. Its only data item is STATE-COUNTRY.

SUB-SCHEMA: That part of the logical data structure set

up for a given application program or programs. Each sub-schema must be described in the schema description. In this case there is only one sub-schema defined. It consists of the entire data base and is named UNIVERSAL. The user has the option of adding others if desired.

TALL-AREA: The name of the area of the data base containing all records except NAME-FILE records.

THIRD-LINE: The 32 character alphanumeric data item within a NAME-FILE record containing the third line of the member's four line address.

TITLE: The 17 character alphanumeric data item within a NAME-FILE record containing the title of the member. If the member is an organizational member, TITLE has the value ORGAN .

UNIVERSAL: The name of the only sub-schema currently defined in the schema description.

ZIP: The 5 character alphanumeric data item making up the last 5 characters of the data item FOURTH-LINE. FOURTH-LINE IS AN ITEM OF THE RECORD TYPE NAME-FILE. If the member has a known zip code, ZIP has the value of that zip code.

APPENDIX C

DOCUMENTATION AND FLOWCHART FOR LOADING PROGRAM

DOCUMENTATION FOR LOADING PROGRAM

The loading program LOADDB.CBL is a COBOL program which loads the data base from two external files. The first, called ROSTER-COM-GROUP, has the internal location numbers, identifiers, and descriptions of the rosters, roster committees, groups, and group committees. It is assumed that the file is structured so that groups are located in the roster that they follow, and that committees are in the roster or group that they follow. The second file, called NAME-ROSTER-FILE, contains information about individual members. The information is whether the member is a person or organization, the name (last name and initials of a person and name of an organization), title, address, state or country, city, organization, and committees, rosters or groups that the member is on.

The program starts by opening the appropriate files and areas of the data base. Since the program changes the data base, its usage mode is exclusive update.

Processing starts by checking the ROSTER-COM-GROUP file to see if there are any more records to process. If not, it branches to the processing of NAME-ROSTER-FILE. ROSTER-COM-GROUP is processed first so that a member may be placed in the appropriate rosters, groups,

and/or committees when NAME-ROSTER-FILE is processed.

The first step in processing a ROSTER-COM-GROUP record is checking the data item TYPE-GROUP. If TYPE-GROUP = "QQROS" PUT-IN-ROSTER is performed. If TYPE-GROUP = "QQCOM" COMMITTEE-CHECK is performed. If TYPE-GROUP is anything else, an error message is displayed and the program reads the next ROSTER-COM-GROUP record.

PUT-IN-ROSTER stores the roster record. RC-NUM is used as both the roster's internal location number and the index for the table LOC-KIND, which keeps the data base key for the ROSTER-REC record (the data base key is used for direct access to the record without going through a calculation procedure) and an indicator to say that it is the key of a ROSTER-REC record. PUT-IN-ROSTER also sets GP-FLAG equal to 0, and calls LEFT-IDENT, which left justifies the roster identification by checking it character by character.

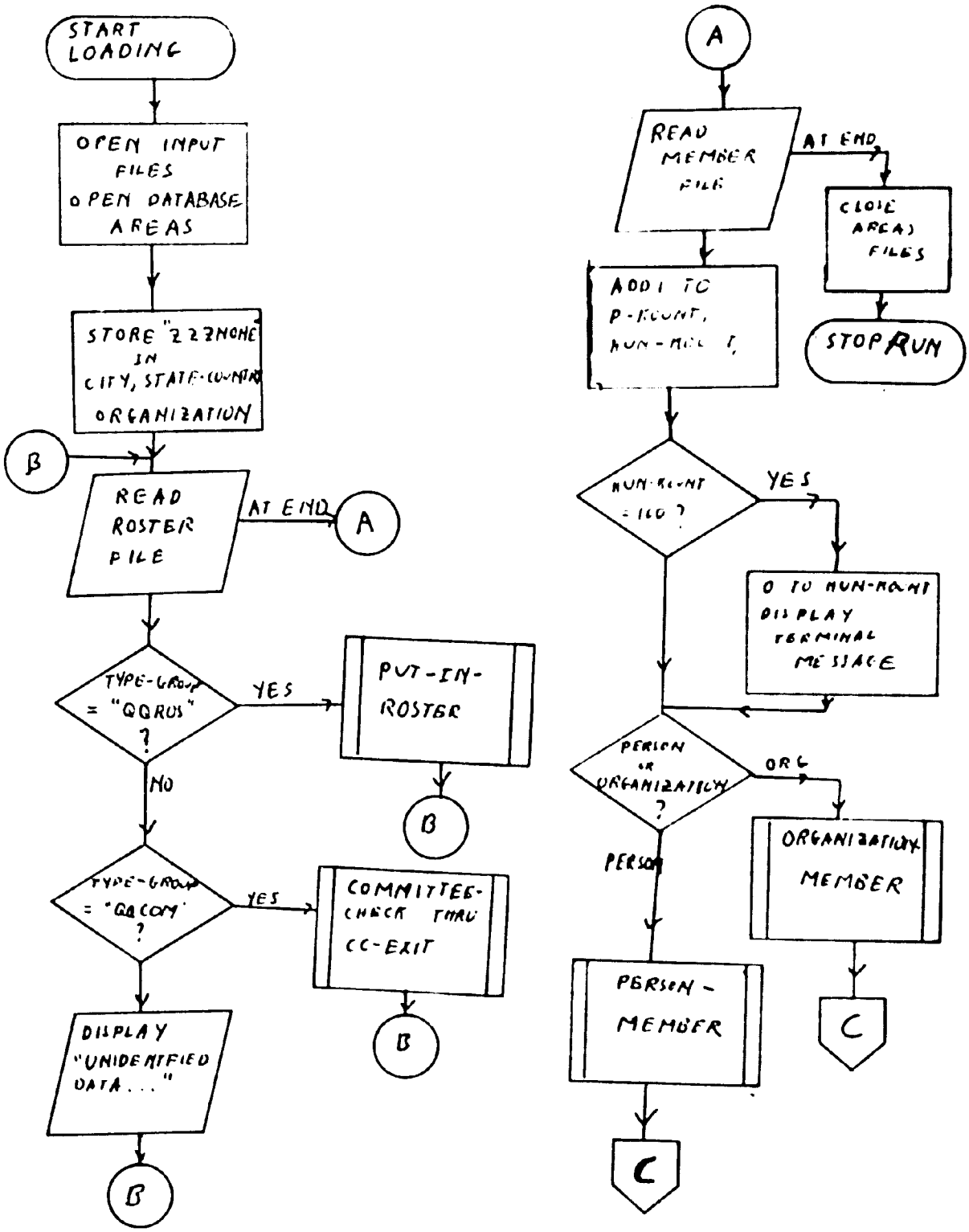
COMMITTEE-CHECK checks to see if TYPE-ID = "GP". If yes, the entity is a group. GP-FLAG is given the value 1, and STORE-GROUP is performed. STORE-GROUP acts similarly to PUT-IN-ROSTER. If TYPE-ID is not "GP", GP-FLAG is checked to see if the entity is a group committee (if GP-FLAG = 1) or a roster committee (if GP-FLAG = 0). Then either STORE-G-COMM or STORE-R-COMM, whichever is appropriate, is performed. Both are similar to

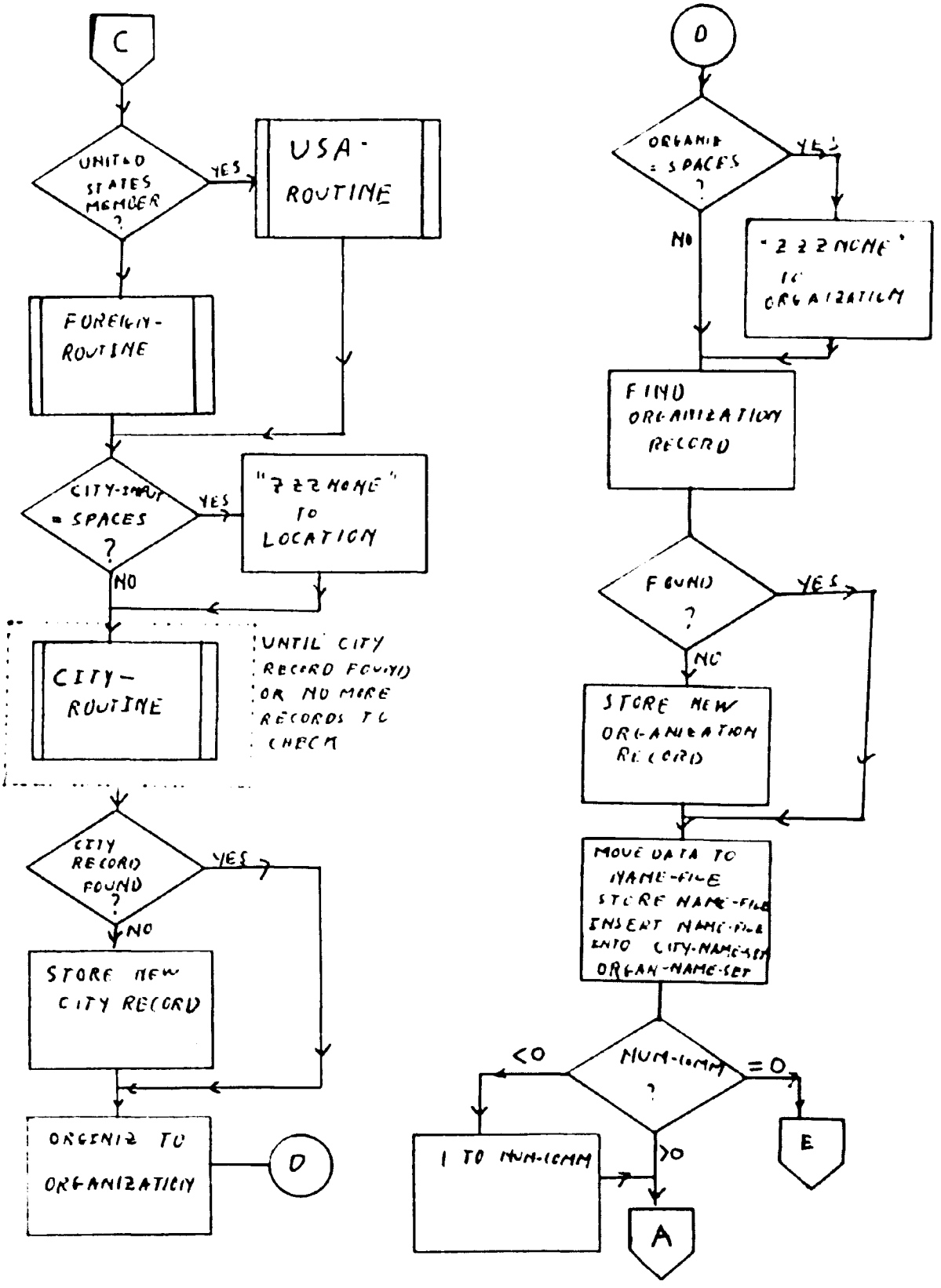
PUT-IN-ROSTER.

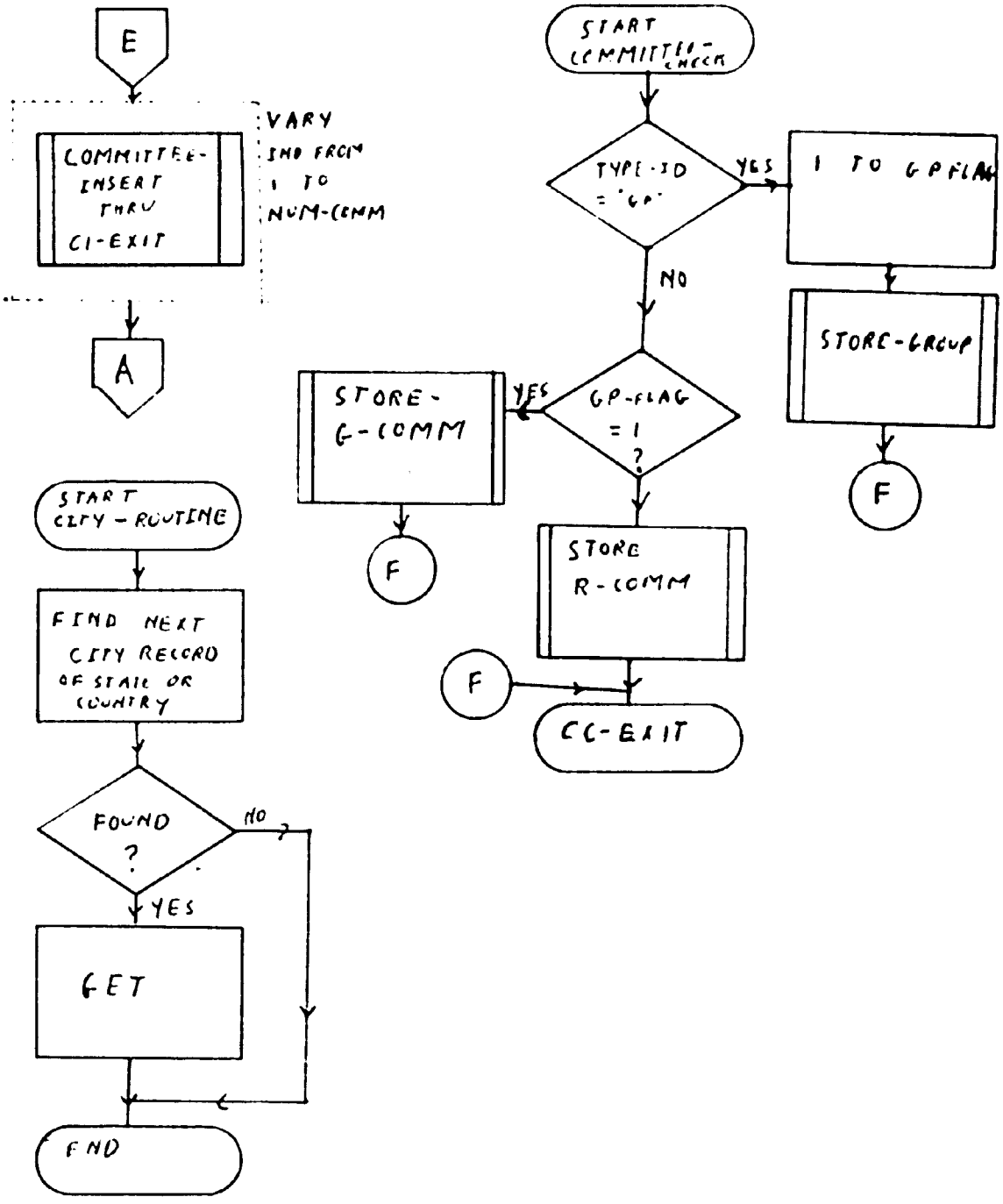
The processing of NAME-ADDRESS-FILE starts by seeing if there are any more records to process. If not, the program ends. If there is a record to process, the first step is to see if the record is for an organizational member or a person member. Then either ORGAN-MEMBER or PERSON-MEMBER is performed to enter the name and title into NAME-FILE. Next the program checks to see if the member is located in the United States. If so, USA-ROUTINE is performed, otherwise FOREIGN-ROUTINE is performed. In either case either the correct STATE-COUNTRY-REC record is found, or a new one is created. Then the program either finds the correct city (using the routine CITY-ROUTINE) or, if the city is not among those listed, creates a new CITY-REC record in the CITY-COUNTRY-SET set occurrence owned by the current STATE-COUNTRY-REC record. The next step the program takes is to either find the correct ORGANIZATION-REC record or create a new one. Then it moves the appropriate data to NAME-FILE, stores NAME-FILE, and inserts it into CITY-NAME-SET set and ORGAN-NAME-SET set.

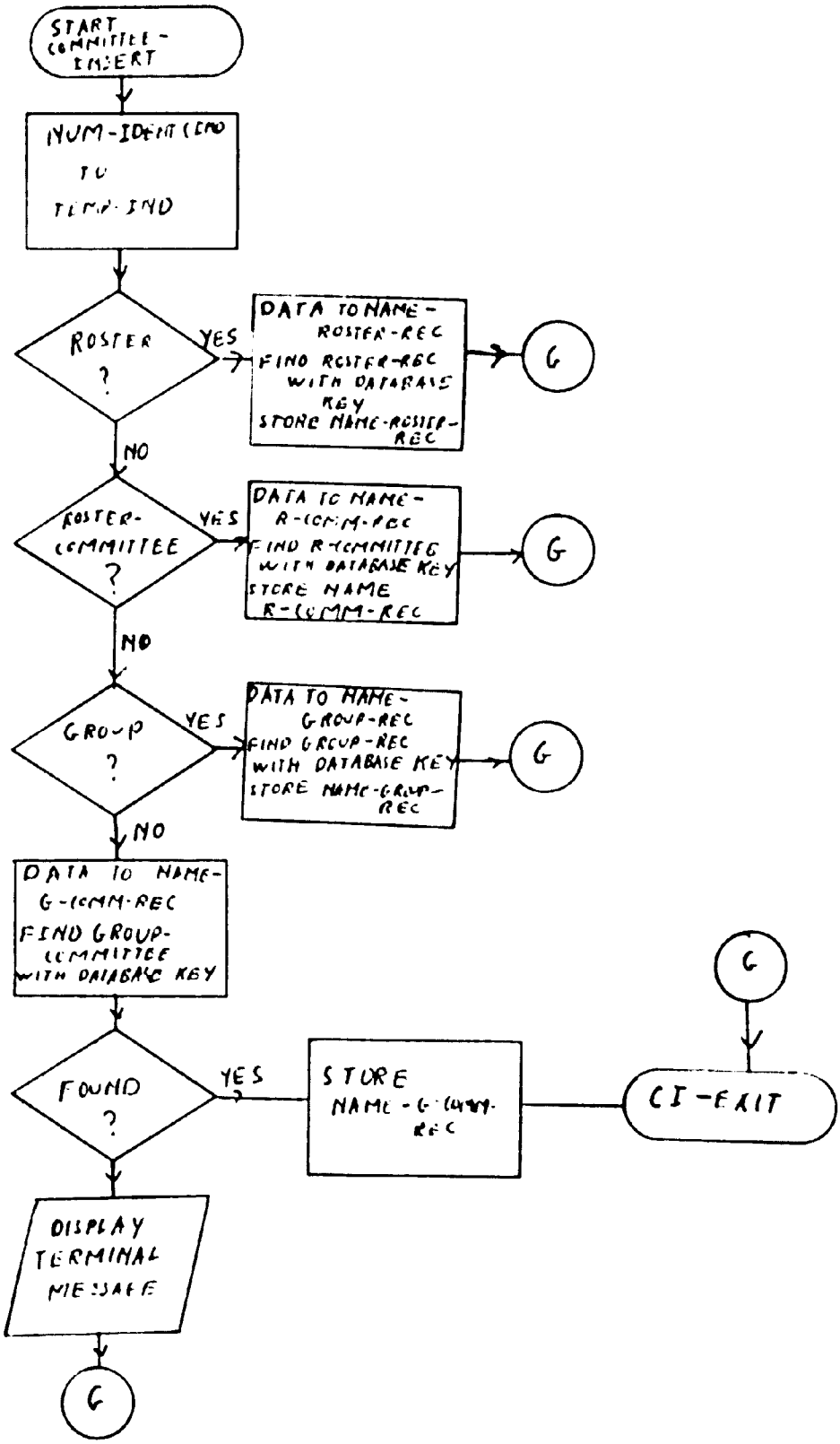
Finally the program checks NUM-COMM to see if the member is on any rosters, committees, and/or groups, and how many the member is on. If NUM-COMM is not zero, the program performs COMMITTEE-INSERT NUM-COMM times.

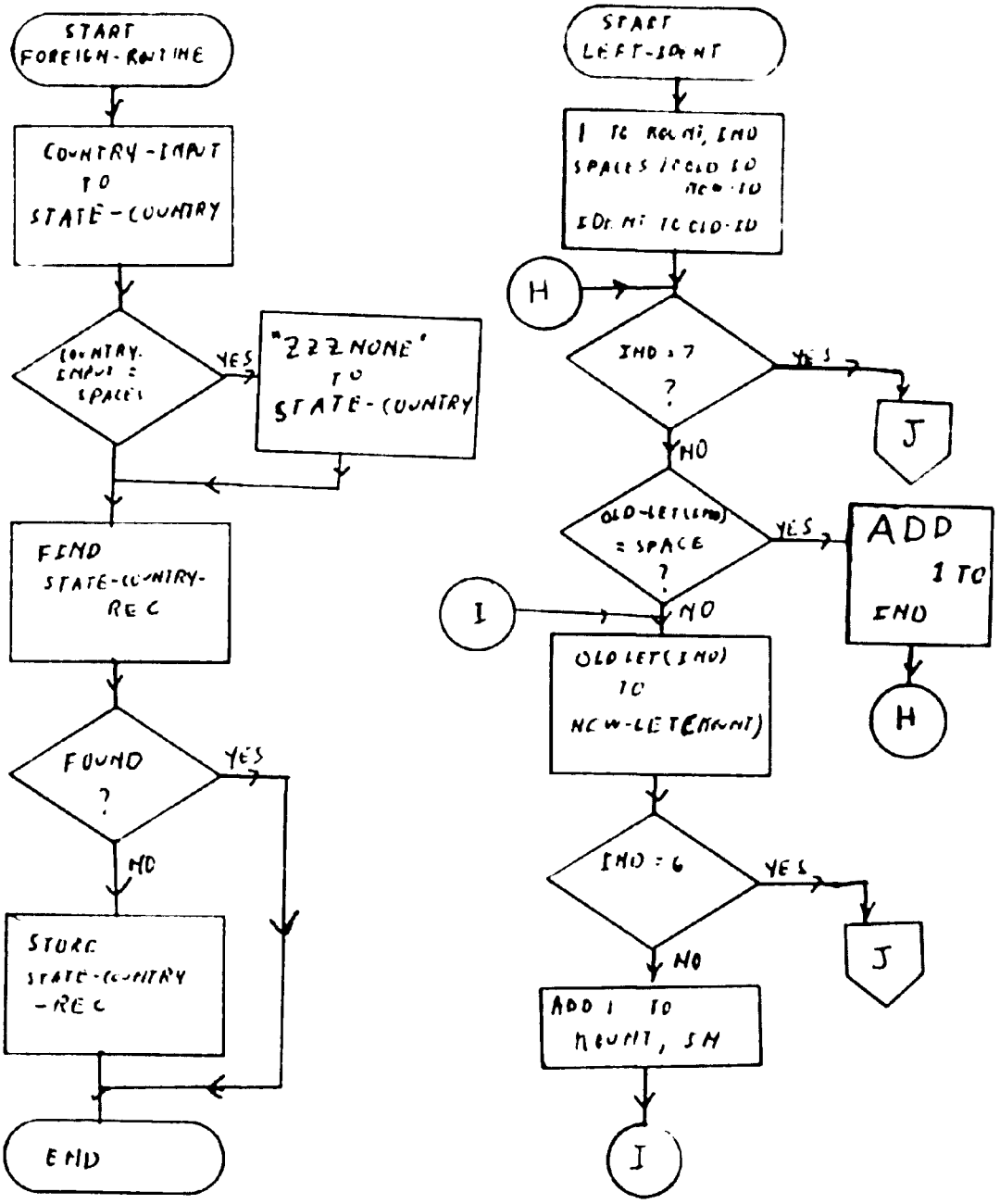
COMMITTEE-INSERT moves the internal location number to a temporary holding place, and used the table LOC-KIND to find the type of entity the member is on, and its data base key. It then puts the member's name and responsibility in the appropriate chain.

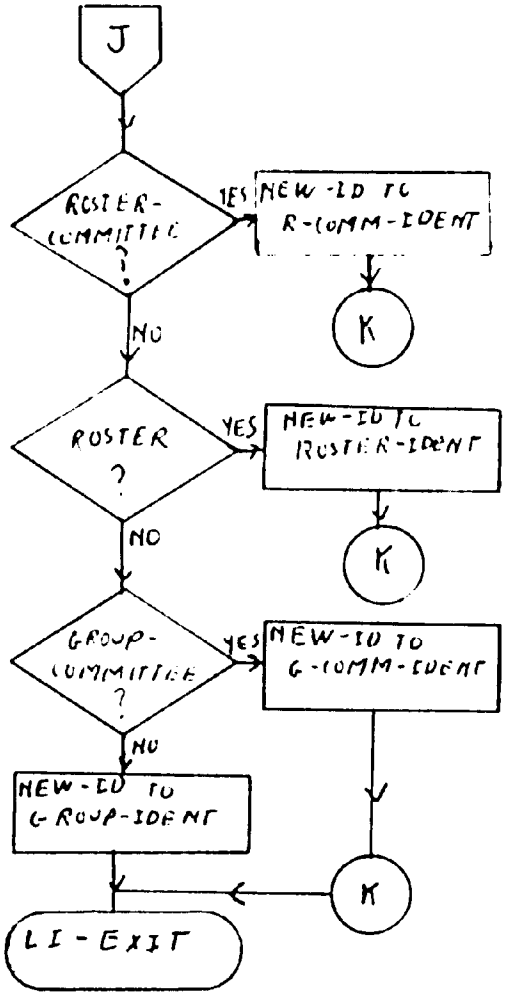

START LOADING
OPEN INPUT FILES
OPEN DATABASE AREAS
STORE "222NONE" IN CITY, STATE-CONTRL ORGANIZATION
B
READ ROSTER FILE
AT END
A
TYPE-GROUP = "QQROS" ?
YES
PUT-IN-ROSTER
B
NO
TYPE-GROUP = "QQCOM" ?
YES
COMMITTEE-CHECK THRU CC-EXIT
B
DISPLAY "UNIDENTIFIED DATA..."
B
A
READ MEMBER FILE
AT END
CLOSE AREAS FILES
STOP RUN
ADD 1 TO P-COUNT, NON-MEC C,
NON-RECNT = 100 ?
YES
0 TO NON-RECNT
DISPLAY TERMINAL MESSAGE
PERSON OR ORGANIZATION ?
ORG
ORGANIZATION MEMBER
C
PERSON
PERSON-MEMBER
C

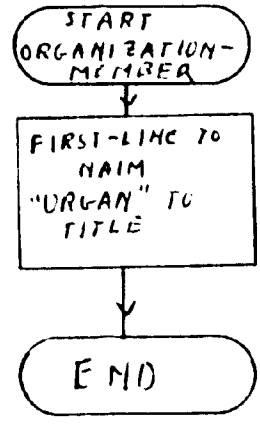

C
UNITED STATES MEMBER ?
YES
USA-ROUTINE
FOREIGN-ROUTINE
CITY-INPUT = SPACES ?
YES
"222 NONE" TO LOCATION
NO
CITY-ROUTINE
UNTIL CITY RECORD FOUND OR NO MORE RECORDS TO CHECK
CITY RECORD FOUND ?
YES
NO
STORE NEW CITY RECORD
ORGINIZ TO ORGANIZATION
D
D
ORGANIZ = SPACES ?
YES
NO
"222 NONE" TO ORGANIZATION
FIND ORGANIZATION RECORD
FOUND ?
YES
NO
STORE NEW ORGANIZATION RECORD
MOVE DATA TO NAME-FILE
STORE NAME-FILE
INSERT NAME-FILE INTO CITY-NAME-SET
ORGAN-NAME-SET
NUM-COMM ?
<0
=0
>0
1 TO NUM-COMM
E
A

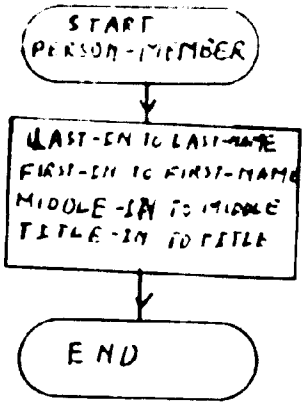

E
COMMITTEE-INSERT THRU CI-EXIT
VARY IND FROM 1 TO NUM-COMM
A
START COMMITTEE-CHECK
TYPE-ID = 'GP'
YES
1 TO GP FLAG
NO
STORE-GROUP
F
GP-FLAG = 1 ?
YES
STORE-G-COMM
F
STORE R-COMM
F
CC-EXIT
START CITY-ROUTINE
FIND NEXT CITY RECORD OF STATE OR COUNTRY
FOUND ?
NO
YES
GET
END

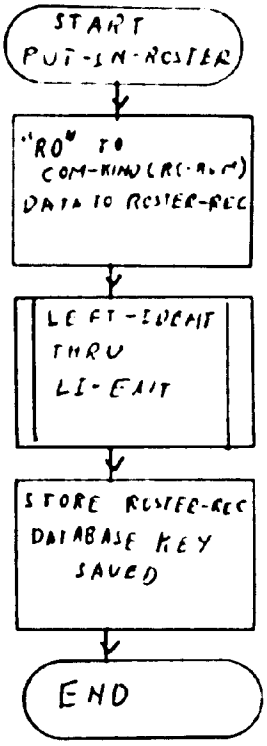

START COMMITTEE-INSERT
NUM-IDENT (END TO TEMP-IND
ROSTER ?
YES
DATA TO NAME-ROSTER-REC
FIND ROSTER-REC WITH DATABASE KEY
STORE NAME-ROSTER-REC
G
NO
ROSTER-COMMITTEE ?
YES
DATA TO NAME-R-COMM-REC
FIND R-COMMITTEE WITH DATABASE KEY
STORE NAME R-COMM-REC
G
NO
GROUP ?
YES
DATA TO NAME-GROUP-REC
FIND GROUP-REC WITH DATABASE KEY
STORE NAME-GROUP-REC
G
NO
DATA TO NAME-G-COMM-REC
FIND GROUP-COMMITTEE WITH DATABASE KEY
FOUND ?
YES
STORE NAME-G-COMM-REC
G
CI-EXIT
DISPLAY TERMINAL MESSAGE
G

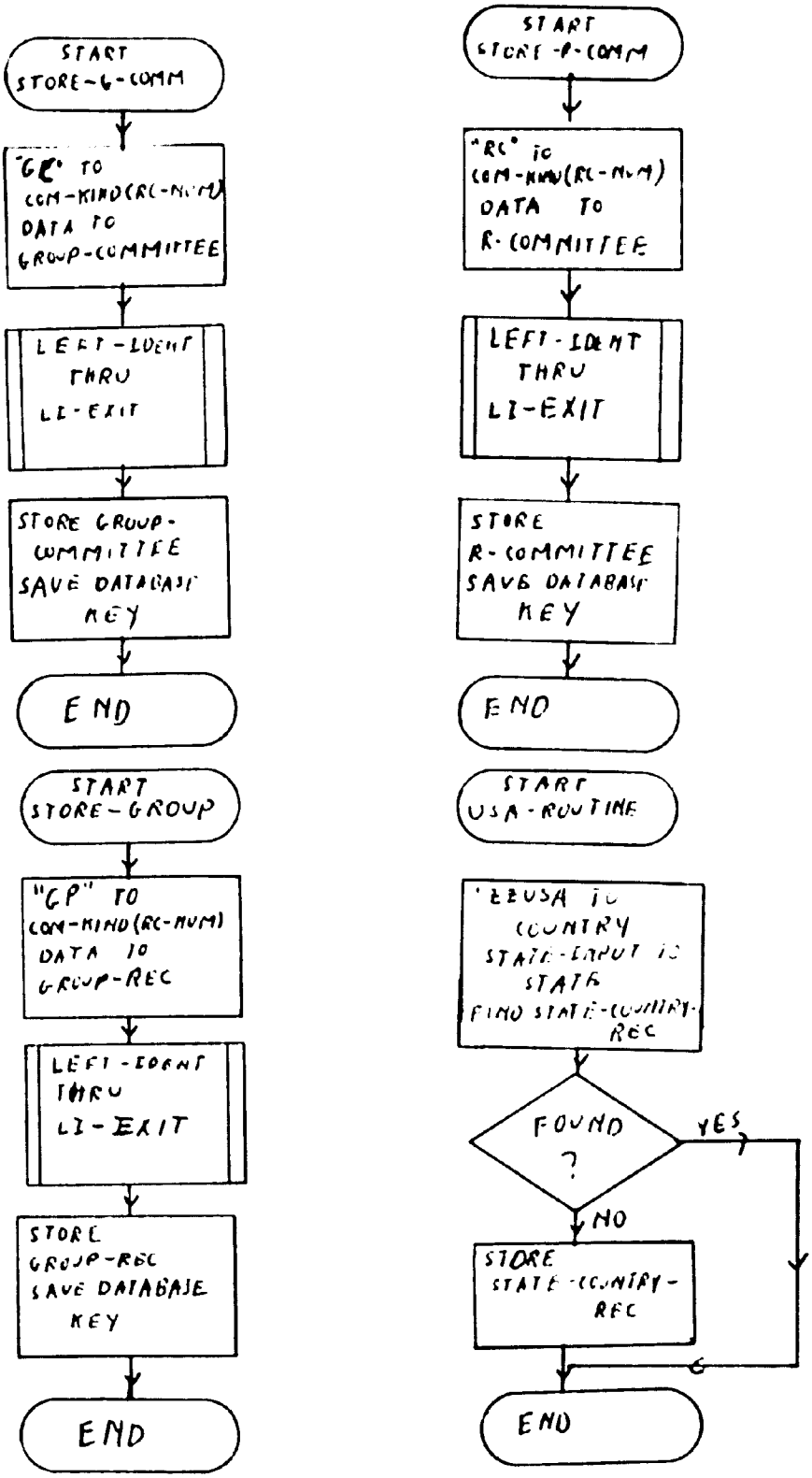

START FOREIGN-ROUTINE
COUNTRY-INPUT TO STATE-COUNTRY
COUNTRY-INPUT = SPACES
YES
"ZZ NONE" TO STATE-COUNTRY
FIND STATE-COUNTRY-REC
FOUND ?
YES
NO
STORE STATE-COUNTRY-REC
END
START LEFT-JOINT
1 TO NUMI, END
SPACES TO OLD-ID NEW-ID
IDX-NI TO OLD-ID
H
END > 7 ?
YES
J
NO
OLD-LET(END) = SPACE ?
YES
ADD 1 TO END
H
I
NO
OLD-LET(END) TO NEW-LET(COUNT)
END = 6
YES
J
NO
ADD 1 TO COUNT, SM
I


J
ROSTER-COMMITTEE ?
YES
NEW-ID TO R-COMM-IDENT
K
NO
ROSTER ?
YES
NEW-ID TO ROSTER-IDENT
K
NO
GROUP-COMMITTEE ?
YES
NEW-ID TO G-COMM-IDENT
NO
NEW-ID TO GROUP-IDENT
K
LI-EXIT


START ORGANIZATION-MEMBER
FIRST-LINE TO NAIM
"ORGAN" TO TITLE
END


START PERSON-MEMBER
LAST-IN TO LAST-NAME
FIRST-IN TO FIRST-NAME
MIDDLE-IN TO MIDDLE
TITLE-IN TO TITLE
END


START PUT-IN-ROSTER
"RO" TO COM-KIND (RC-ROOM)
DATA TO ROSTER-REC
LEFT-IDENT THRU LI-EXIT
STORE ROSTER-REC
DATABASE KEY SAVED
END


START
STORE-G-COMM
"GC" TO
COM-KIND(RC-NUM)
DATA TO
GROUP-COMMITTEE
LEFT-IDENT
THRU
LI-EXIT
STORE GROUP-
COMMITTEE
SAVE DATABASE
KEY
END
START
STORE-GROUP
"GP" TO
COM-KIND(RC-NUM)
DATA TO
GROUP-REC
LEFT-IDENT
THRU
LI-EXIT
STORE
GROUP-REC
SAVE DATABASE
KEY
END
START
STORE-R-COMM
"RC" TO
COM-KIND(RC-NUM)
DATA TO
R-COMMITTEE
LEFT-IDENT
THRU
LI-EXIT
STORE
R-COMMITTEE
SAVE DATABASE
KEY
END
START
USA-ROUTINE
"EZUSA" TO
COUNTRY
STATE-INPUT TO
STATE
FIND STATE-COUNTRY-
REC
FOUND
?
YES
NO
STORE
STATE-COUNTRY-
REC
END

APPENDIX D

DOCUMENTATION AND FLOWCHART

FOR UPDATE PROGRAM

DOCUMENTATION FOR UPDATE PROGRAM

The program UPDB.CBL is a COBOL program intended for on-line updating of the data base. It can add a new record for a member (either a person or an organizational member), roster, group, or committee. It can modify any field in the record of an existing member, roster, group, or committee, with the exception of the field used for the internal location number of a roster, group, or committee. It can delete the record of a member, roster, group, or committee.

The program runs under the exclusive update mode. As the program runs it creates separate disc files giving the additions, deletions, and changes that were made during a particular run. The files are ADDED.DAT, CHANGE.DAT, and DELETE.DAT.

The opening paragraph of the procedure division opens the output files and the areas of the data base, and performs HEADER-ROUTINE (line 97000) which writes the headings of the output reports.

The beginning of every group of transactions, i.e., transactions involving one particular entity, starts with the paragraph WHICH-UPDATE (line 10500). WHICH-UPDATE starts by displaying a message explaining the types of

updates available to the user, and asks the user to enter the character which identifies the current choice. An available option is to enter E to end execution of the program. The routine QUESTION-ANSWER (line 99880) is then performed. QUESTION-ANSWER is used whenever the user is to input a one-character item from the terminal. The routine moves the item to the data field TYPE-UPDATE, which is then checked for the value of the item. In this case TYPE-UPDATE is first checked to see if the item is a valid one. If not, a message to that effect is displayed, and control goes to the start of WHICH-UPDATE. If three successive illegal entries are made, the program ceases execution.

If the input character is valid, and not E, the program branches to either ADD-NEW (line 12900), CHANGE-OLD (line 39100), or DELETE-OLD (line 62500) depending on whether the update is an addition, modification, or deletion. In each case the user specifies one of three possibilities: the update is for a person member; the update is for an organizational member; or the update is for a roster, group or committee. Thus there are nine major independent branches in the program. They are independent in the sense that the program never branches from one to another except for the purpose of executing particular lines of code via PERFORM commands. This

documentation will examine the three types of additions, followed by the three types of modifications, followed by the three types of deletions. Descriptions of performed routines which are not in one of the nine major branches are in alphabetical order following the descriptions of the major branches.

NEW-PERSON (line 14600):

This paragraph starts the code used for adding a new person member to the data base. It accepts the last name of the person, calls the routine LEFT-JUST THRU LJ-EXIT to left justify the terminal input (all alphabetic terminal input is left justified with this routine) and moves it to LAST-NAME. The paragraph FIRST-AND-MIDDLE then does the same for the first name and the middle name. TITLE-CHANGE receives the title of the person and ADDRESS-ROUTINE the last three lines of the label address. The address is not checked for content, but merely moved to the appropriate fields of NAME-FILE. USA is used to find if the new member is located in the United States or not. Then NEW-USA THRU ZIP-ROUTINE (line 69700) or NEW-FOREIGN (line 75800) whichever is appropriate is performed. These routines either find or create records for the associated cities and states or countries. PHONE-ROUTINE and START-DATE-ROUTINE accept and place in NAME-FILE the phone number

and starting date with the project, respectively, of the new member. NEW-ORG-ROUTINE either finds or creates the correct ORGANIZATION-REC record for the member. If there is no organization, the individual is associated with the record where ORGANIZATION has the value ZZZNONE . The lines of code referred to above, i.e., the paragraphs FIRST-AND-MIDDLE, TITLE-CHANGE, ADDRESS-ROUTINE, USA, PHONE-ROUTINE, START-DATE-ROUTINE, and NEW-ORG-ROUTINE, have separate paragraph names because the same code is used in other major branches, and is called by PERFORM statements.

PLACE-INDIVIDUAL puts the member's record into the data base. It stores NAME-FILE, connects it to the appropriate city and organization, and calls CONNECT-ROSTER THRU CR-EXIT (line 79100) which puts the member on the appropriate roster, committee, and/or group lists. Then PLACE-INDIVIDUAL writes the output report and returns control to WHICH-UPDATE.

NEW-ORG (line 25000):

This paragraph starts the branch of the program used to create a new organization member, and is very similar to the branch starting with NEW-PERSON. An organizational member has its name entered as a whole, and the value ORGAN is given to TITLE. After accepting the name, the program moves it to NAIM and ORGANIZATION

and checks to see if the organization is already listed as an occurrence of the record type ORGANIZATION-REC. If not, a new ORGANIZATION-REC record is created. Then the program performs ADDRESS-ROUTINE THRU START-DATE-ROUTINE (see NEW-PERSON), stores NAME-FILE, connects it to the appropriate CITY-REC and ORGANIZATION-REC records, peforms CONNECT-ROSTER THRU CR-EXIT (line 79100), writes the output report, and returns control to WHICH-UPDATE.

NEW-ROSTER (line 26800):

This paragraph starts the branch used to create a record for a new roster, group, or committee. It starts by accepting the number used for internal identification, and performs NUM-RECEIVE THRU NR-EXIT (line 99605) which moves the input to the data item NEW-ROS-NUM. The program then checks to see if the type of input is valid, i.e., numeric and three digits or less. If the input is valid NEW-NUM-CHECK THRU NNC-EXIT (line 39005) is performed to see if the identifying number is already being used. If so, a message to that effect is displayed, and control is transferred back to WHICH-UPDATE. If the number is not in use already, the program successively accepts the type, identifier, output code, and description.

If TYPE-UPDATE is R the program stores ROSTER-REC, performs ROS-COM-ADD-OUTPUT (line 37300), and

transfers control to WHICH-UPDATE. If TYPE-UPDATE is the G the program performs STORE-NEW-GROUP (line 37700), which either finds the record of the roster that the new group is in, or returns an indicator saying that it cannot be found. Then, if there were no problems, GROUP-REC is stored, ROS-COM-ADD-OUTPUT is performed, and control is returned to WHICH-UPDATE.

If TYPE-UPDATE is C, execution continues at ASSOCIATED-ROSTER with the program asking for the number of the associated roster and then trying to find its record. If a ROSTER-REC record with the input number cannot be found, the program looks for a GROUP-REC record with that number. If one is found, the new entity is stored as a group committee within that group, ROS-COM-ADD-OUTPUT is performed, and control is returned to WHICH-UPDATE. If a GROUP-REC record is not found, an error message is displayed and control is transferred back to ASSOCIATED-ROSTER. A maximum of three consecutive incorrect inputs is permitted before the program ceases execution.

If a valid roster number is input, the program performs FIND-GROUP (line 78700) to see if the roster is divided into groups. If not R-COMMITTEE is stored, ROS-COM-ADD-OUTPUT is performed, and control is transferred back to WHICH-UPDATE. If the roster is divided into

groups, the program asks if the user wishes to have the committee connected to a group. If not, R-COMMITTEE is stored as above. If the user does so wish, the program asks for the correct group number. Three chances are given for valid input. If valid input is received GROUP-COMMITTEE is stored, ROS-COM-ADD-OUTPUT is performed, and control is transferred to WHICH-UPDATE.

PEOPLE-CHANGE (line 41000):

This paragraph starts the branch which alters the record of a person member. It starts by performing LAST-NAME-ONLY THRU LNO-EXIT (line 68302), which asks if the user wishes to enter only the last name or the entire name. The data item NAME-FOUND is returned. If it is 2, then there is no record to be changed, and the program goes to WHICH-UPDATE. If NAME-FOUND is one, the record has been found, and the program branches to paragraph START-CHANGE (line 43100). If NAME-FOUND is 0, that means the user wishes to enter the entire name. It starts off by accepting the last name, first name, and middle name of the person. The main difference between this start and the start of NEW-PERSON is that the message displayed at the start here emphasizes the necessity of entering the name exactly as it is currently recorded. After accepting the last name, the first and middle names are accepted by performing FIRST-AND-MIDDLE (see NEW-

PERSON). FIND-RIGHT-DUPLICATE (line 99930) is then performed until either the record of the correct person is found or there are no more records of people whose name is the same as that entered. In the latter case an appropriate message is displayed, and control reverts to WHICH-UPDATE.

When the desired record is found, the program saves the data base key of the record, and then asks a series of questions where the user enters Y to indicate an affirmative answer, and anything else to indicate a negative answer. Before each question is asked an appropriate message is displayed, then the question is asked, and then QUESTION-ANSWER is performed. In order the questions are:

Change name? If yes, perform NEW-PERSON THRU FIRST-AND-MIDDLE, then perform NAME-ROS-CHANGE THRU NRC-EXIT (line 94100), which makes the same name change on roster, committee, and group lists. Then the correct NAME-FILE record is found with the data base key to keep it the current record of the run-unit.

Change title? If yes, perform TITLE-CHANGE (see NEW-PERSON).

The paragraph ADDRESS-EXPLAN is started.

Change address? If yes, perform ADDRESS-ROUTINE (see NEW-PERSON).

Change phone number? If yes, perform PHONE-ROUTINE (see NEW-PERSON).

Change starting date? If yes, perform START-DATE-ROUTINE (see NEW-PERSON).

Enter an end date? If yes, perform END-DATE-ROUTINE (line 85190).

Then NAME-FILE is modified, and the paragraph COUNTRY-STATE-CHANGE is started. The user is asked if the country or state is to be changed. If not, the program branches to CITY-CHANGE. If yes, the program finds the current city (if there is one) and removes the NAME-FILE record from the CITY-NAME-SET set occurrence it is in. The program then asks if the new country is the United States. If yes, NEW-USA (line 68700) is performed, otherwise NEW-FOREIGN (line 75800) is performed. NAME-FILE is then inserted into CITY-NAME-SET set and the program branches to ORGAN-CHANGE.

CITY-CHANGE starts by asking if the city is to be changed. If not, the program branches to ORGAN-CHANGE. If yes, the program tries to find the record of the current city. If the record cannot be found, NO-CITY-CHANGE (line 62005) is performed, and the program branches to ORGAN-CHANGE. If there is a record of the current city, the program finds the current state or country record, disconnects the member's record from the

current city, performs NEW-CITY THRU NC-EXIT (line 74500), and finds the current NAME-FILE record via the data base key. The member's record is then connected to the new city's record.

ORGAN-CHANGE asks if the member's organization is to be changed. If not, the program branches to ROSTER-CHANGE. Otherwise the program removes the member's name from the current organization's list (if there is a current organization), and connects the name to the new organization by performing NEW-ORG-ROUTINE (see NEW-PERSON).

ROSTER-CHANGE starts by asking if the committee or roster affiliations are to be changed. If not, the output report is written, and control transfers to WHICH-UPDATE. If yes, then a message is displayed which says that deletions, additions, and changes in responsibility will be handled in that order. Then paragraph DELETE-ROS starts.

DELETE-ROS asks for a number of a roster or committee (or group) where the member's name is to be deleted from the list. ACCEPT-ROS-NUM THRU ARN-EXIT (line 86300), which accepts the input number and identifies the associated entity, is performed. If spaces were entered, the program branches to ADD-ROS-CHANGE. If invalid input is entered, an error message is displayed

and DELETE-ROS starts again. A maximum of three consecutive invalid inputs are permitted. If a valid input is entered, the program checks to see whether it is for a roster, roster committee, group, or group committee, and performs either DEL-ROS-NAME, DEL-R-COMM-NAME, DEL-GROUP-NAME, or DEL-G-COMM-NAME (line 91200). Then it goes back to DELETE-ROS.

ADD-ROS-CHANGE performs BEGIN-ROS THRU CR-EXIT (see CONNECT-ROSTER THRU CR-EXIT, line 79100) to add the member's name to the desired rosters, groups, and/or committees.

CHANGE-RESPONS acts similarly to DELETE-ROS. It uses ACCEPT-ROS-NUM THRU ARN-EXIT to accept and check the identifying number. If a valid number is input, CHANGE-RESPONS accepts the new responsibility and performs either CHANGE-ROSTER-RESPONS, CHANGE-R-COMM-RESPONS, CHANGE-GROUP-RESPONS, or CHANGE-G-COMM-RESPONS (line 88800). When there are no more inputs the output report is written, and control reverts to WHICH-UPDATE.

ORGANIZATION-CHANGE (line 55800):

This routine accepts the name of the organization, finds the appropriate NAME-FILE record (if possible), displays the name and the address, and asks if the displayed organization is the desired one.

If not, it looks for others until either the desired one is found, or there are no more organizations with the input name. In the latter case a message to that effect is displayed, and control is transferred back to WHICH-UPDATE.

When the correct record is found, the program asks if the user wishes to change the name. If not, the program branches to NEXT-QUESTION. If yes, the program looks for the associated ORGANIZATION-REC record, and if there is none, one is created. The new name is accepted, moved to NAIM and ORGANIZATION, NAME-ROS-CHANGE THRU NRC-EXIT (line 94100) is performed, and ORGANIZATION-REC is modified.

NEXT-QUESTION starts by performing ADDRESS-EXPLAN (see PEOPLE-CHANGE). Then the program asks if the country or state is to be changed. If not, the program branches to CITY-O-QUESTION. Otherwise it removes the member's record from the city list it is on, performs either NEW-USA (line 69700) or NEW-FOREIGN (line 75800), and puts the member's record on the new city list. ORG-ROS-CHANGE (line 62410), which takes care of roster and committee changes, is performed, CHANGE-OUTPUT-REPORT (line 62100), which writes the output report, is performed, and control is transferred to WHICH-UPDATE.

person member. It starts by performing LAST-NAME-ONLY THRU LNO-EXIT (line 68302). If the data item NAME-FOUND is 2, there is no record of the person, and the program goes back to WHICH-UPDATE. If NAME-FOUND is 1, the record has been found, and is deleted. If NAME-FOUND is 0, the program continues by performing NEW-PERSON THRU FIRST-AND-MIDDLE. FIND-RIGHT-DUPLICATE THRU FRD-EXIT (line 99930) is performed until either the correct record is found or there are no more records of people whose name is the same as that input. Then either the record is deleted and DELETE-REPORT-ROUTINE (line 67600) is performed, or a message indicating the record could not be found is displayed. Then control passes back to WHICH-UPDATE.

ORGANIZATION-OUT (line 64700):

This works exactly as PEOPLE-OUT, except ORGANIZATION-CHANGE is performed at the start.

ROS-COM-OUT (line 65200):

This branch starts by asking for the identifying number, and then performing ACCEPT-ROS-NUM THRU ARN-EXIT (line 86300). Then if a valid number is entered, the appropriate entity is deleted, DELETION-REPORT-ROUTINE is performed, and control is transferred to WHICH-UPDATE.

CITY-O-QUESTION is used to check for city changes when the state or country is not changed. Its logic is very similar to CITY-CHANGE in the PEOPLE-CHANGE branch of the program. The main difference is that where CITY-CHANGE branches to ORGAN-CHANGE, CITY-O-QUESTION performs ORG-ROS-CHANGE, CHANGE-OUTPUT-REPORT, and branches to WHICH-UPDATE.

CHANGE-CODE (line 84685):

This branch is to change one or more fields in the record of a roster, group or committee. It starts by asking for the identifying number, and performs ACCEPT-ROS-NUM THRU ARN-EXIT (line 86300) to check the validity of the input, find what type of entity the number identifies, and make the correct record the current record of the run-unit. If the input is valid CHANGE-CODE gets the record and asks in turn if the user wishes to change the type, identifier, description, and output code. If at any time the answer is no, it branches down to the next question, otherwise it moves the new data item to the appropriate fields of all four types of entities. At the end it modifies the current record, writes the output report, and transfers control to WHICH-UPDATE.

PEOPLE-OUT (line 64100):

This branch is used to delete the record of a

ALPHABETICAL LIST OF ROUTINES WHICH ARE NOT CONTAINED WITHIN ONE OF THE NINE MAJOR LOGICAL BRANCHES

ACCEPT-ROS-NUM THRU ARN-EXIT (line 86300):

This routine accepts an identifying number, checks its validity, and (if the number was valid) finds the record of the appropriate entity. It moves an indicator to the data item R-FLAGG to tell whether the input was spaces, not valid, or the type of entity if the input number was valid.

ADDITION-OUTPUT-REPORT (line 78605):

This moves and writes the name and address lines of a new member on the addition report.

CHANGE-G-COMM-RESPONS (line 90000):

This performs FIND-G-COMM-NAME (line 93800) until either the appropriate name is found or there are no more names to check. Then if the name was found, the responsibility is modified. Otherwise an error message is displayed.

CHANGE-GROUP-RESPONS (line 90600):

Almost identical to CHANGE-G-COMM-RESPONS.

CHANGE-OUTPUT-REPORT (line 62100):

This moves the name to the output line, writes the change report from the output line, and then moves spaces to the output line.

CHANGE-R-COMM-RESPONS (line 89400):

Almost identical to CHANGE-G-COMM-RESPONS.

CHANGE-ROSTER-RESPONS (line 88800):

Almost identical to CHANGE-G-COMM-RESPONS.

CONNECT-ROSTER THRU CR-EXIT (line 79100):

This routine is to put a member on lists of rosters, groups, and/or committees. It starts with a message to that effect. The actual process starts with paragraph BEGIN-ROS.

BEGIN-ROS asks for and accepts the identifying number, and then performs NUM-RECEIVE THRU NR-EXIT (line 99605) to get the number to the data item NEW-ROS-NUM. If the input is valid an appropriate identifier is moved to R-FLAGG, and the program branches to ADD-RESPONS. Otherwise an error message is displayed, and the program goes back to BEGIN-ROS. If the input is blank, the routine ends.

ADD-RESPONS accepts the responsibility, checks R-FLAGG for the appropriate type of entity, moves the data to either ROSTER-RESPONS, R-COMM-RESPONS, GROUP-RESPONS, or G-COMM-RESPONS, depending on R-FLAGG, and stores either NAME-ROSTER-REC, NAME-R-COMM-REC, NAME-GROUP-REC, or NAME-G-COMM-RESPONS. Then it branches back to BEGIN-ROS.

DEL-G-COMM-NAME (line 93400):

This performs FIND-G-COMM-NAME (line 93800) until either the appropriate name is found, or there are no more names. If the name is found NAME-G-COMM-REC is deleted.

DEL-GROUP-NAME (line 93100):

Acts the same as DEL-G-COMM-NAME.

DEL-R-COMM-NAME (line 92400):

Acts the same as DEL-G-COMM-NAME.

DEL-ROS-NAME (line 91200):

Acts the same as DEL-G-COMM-NAME.

DELETION-REPORT-ROUTINE (line 67600):

This moves the report material to PRINT-LINE, writes the deletion report, and then moves spaces to PRINT-LINE.

END-DATE-ROUTINE (line 85190):

This routine displays a message explaining how to enter an end date, asks for and accepts the date, performs LEFT-JUST THRU LJ-EXIT (line 68400), moves the data to END-DATE, and replaces spaces by zeros.

FIND-AND-PUT-IN (line 96920):

This tries to find the desired STATE-COUNTRY-REC record. If it cannot be found, the routine creates a new STATE-COUNTRY-REC record, and moves 1 to the item

NOSTATE as an indicator that it did so.

FIND-CITY (line 74300):

This finds the next city belonging to a particular state or country, and then gets the record if the find was successful.

FIND-G-COMM-NAME (line 93800):

This finds the next NAME-G-COMM-REC record of a particular group committee, and if successful, gets the record.

FIND-GROUP (line 78700):

This is to find if a particular roster has groups. If it does, 1 is moved to FOUND-GROUP as an indicator.

FIND-GROUP-NAME (line 93100):

Acts the same as FIND-G-COMM-NAME.

FIND-R-COMM-NAME (line 92400):

Acts the same as FIND-G-COMM-NAME.

FIND-RIGHT-DUPLICATE THRU FRD-EXIT (line 99930):

If the item TEMP-IND has the value zero, the routine just finds NAME-FILE record. If there is none to find, the routine stops. If there is a record, the routine gets it, displays the name and address, and asks if the record is the one desired.

If TEMP-IND is not zero, indicating this is a

retry, the routine looks for the next record which duplicates the input name, and acts as described in the previous paragraph.

FIND-ROSTER-NAME (line 91600):

Acts the same as FIND-G-COMM-NAME.

HEADER-ROUTINE (line 97000):

This routine writes the headings for the three output reports.

LAST-NAME-ONLY (line 68302):

This routine asks if the user wishes to enter only the last name of the member whose record is to be found. If not, the routine ends. If so, NEXT-LAST-NAME THRU NLN-EXIT (line 68378) is performed to find the correct record, after the routine has accepted the last name, and created a dummy record. If the name is found, 1 is moved to NAME-FOUND, otherwise 2 is moved to NAME-FOUND. Then the dummy record is deleted.

LEFT-JUST THRU LJ-EXIT (line 68400):

This routine tests the item NEW-NAME character by character. When it finds the first non-blank character, it moves STOP-NUM successive characters, starting with the first non-blank character, from NEW-NAME to NEW-LEFT, where they will be left justified.

NAME-ROS-CHANGE THRU NRC-EXIT (line 94100):

If the name of a member was changed during a run using either the branch PEOPLE-CHANGE or the branch ORGANIZATION-CHANGE, this routine is used to find and modify all the associated NAME-ROSTER-REC, NAME-G-COMM-REC, NAME-R-COMM-REC, and NAME-GROUP-REC records.

NEW-CITY THRU NC-EXIT (line 74500):

This routine accepts the name of a city. Then if NOSTATE has the value 1, indicating it is a city in a state or country whose record was just created, the routine creates a new CITY-REC record for the city. If NOSTATE has any other value, the routine performs FIND-CITY (line 74300) until the appropriate CITY-REC record is found, or there is no CITY-REC record for the input city. In the latter case NO-STATE (line 72100) is performed.

NEW-FOREIGN (line 75800):

This routine asks for and accepts the name of a country, and tries to find the associated STATE-COUNTRY-REC record. If the record is not found NO-STATE (line 72100) is performed. Then NEW-CITY THRU NC-EXIT (line 74500) is performed.

NEW-NUM-CHECK THRU NNC-EXIT (line 39005):

This routine is used to check an input identifying number for a new roster, group, or committee to

see if the number is already being used. If the number is being used, BN is moved to R-FLAGG.

NEW-PERSON-ORG (line 77900):

When a record for a new person is added to the data base, or when the organization of a person with an existing record is changed, this routine tries to find the record of the new organization. If the record cannot be found, a new ORGANIZATION-REC record is created.

NEW-START (line 77200):

This routine moves the starting date to START-DATE and replaces spaces by zeros.

NEW-USA THRU ZIP-ROUTINE (line 69700):

This routine moves ZZUSA to COUNTRY, asks for and accepts the name of the state, and moves it to STATE. Then it tries to find the associated STATE-COUNTRY-REC record. If the record is not found, NO-STATE (line 72100) is performed. Then NEW-CITY THRU NC-EXIT is performed.

ZIP-ROUTINE asks for and accepts the zip code, and moves it to ZIP.

NEXT-LAST-NAME THRU NLN-EXIT (line 68378):

This routine starts with a dummy record and finds the next NAME-FILE record. Then it displays the name and address of the member. This process is repeated

until either the last name of the member does not match the desired last name (stored in data item TEMP-NAME), or the user indicates that the correct record has been found.

NO-CITY-CHANGE (line 62005):

This routine is used when a member's city is to be changed and there is no current city. The routine asks if the city is in the United States. If so, NEW-USA (line 69700) is performed, otherwise NEW-FOREIGN (line 75800) is performed. Then the member's NAME-FILE record is linked to the record of the new city.

NO-STATE (line 72100):

This routine is used as a check against a misspelled state, city, or country. It asks for and accepts the new entity. If the entity is either a state or country FIND-AND-PUT-IN (line 96920) is performed. If the entity is a city FIND-CITY (line 74300) is performed until the record of the city is found, or there are no more records to check. In the latter case a new CITY-REC record is stored.

NUM-RECEIVE THRU NR-EXIT (line 99605):

This routine is used to move an identifying number for a roster, group, or committee to the data item NEW-ROS-NUM. The input must be numeric and three digits or less. Should either of these conditions be

violated, B is moved to the item BAD-INPUT. If the input is valid, it is right justified in NEW-ROS-NUM.

ORG-ROS-CHANGE (line 62410):

This routine asks if the roster affiliations of an organizational member are to be changed. If the answer is yes, DELETE-ROS THRU ADD-ROS-CHANGE (see PEOPLE-CHANGE) is performed.

QUESTION-ANSWER (line 99880):

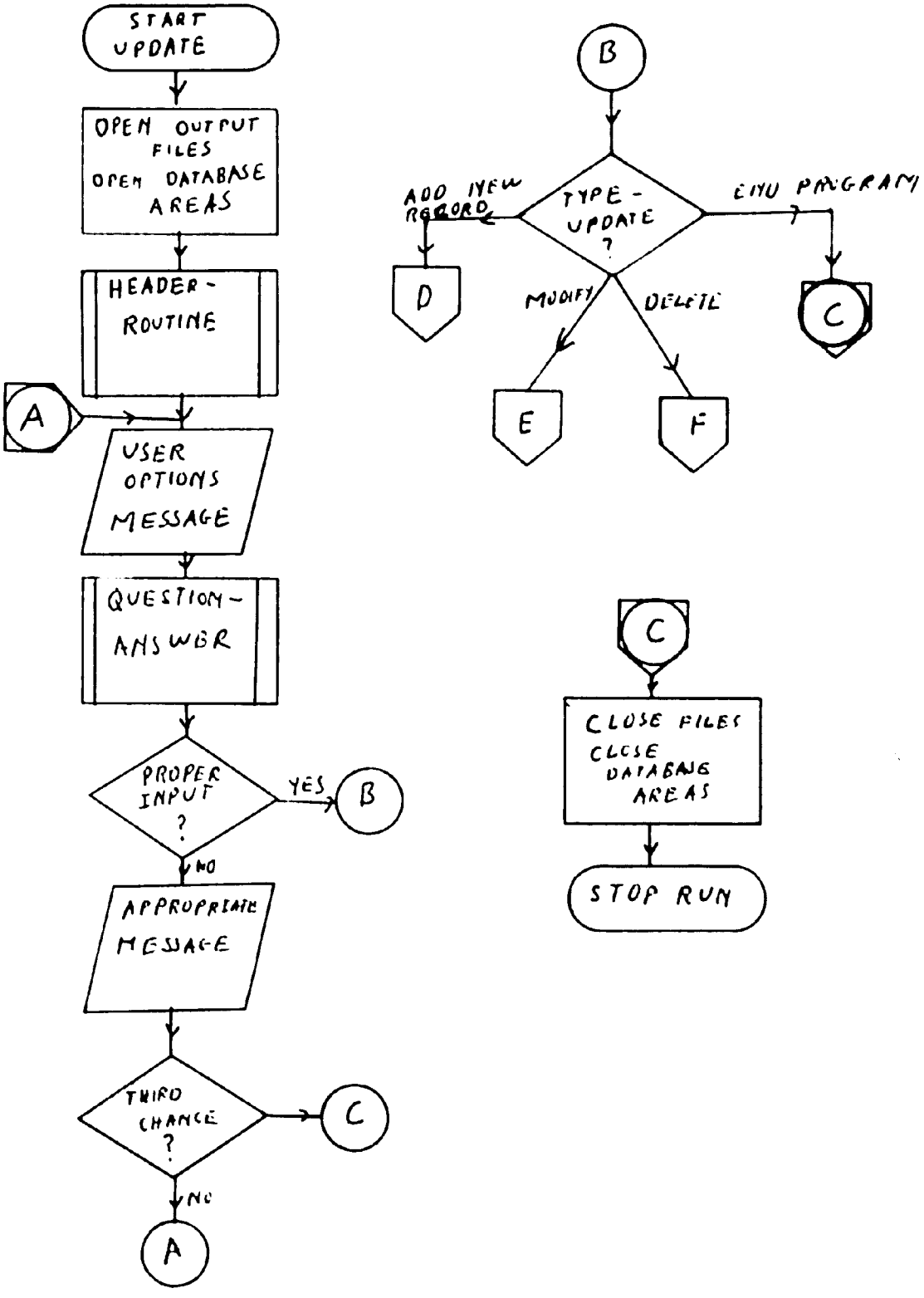
This routine is used to move a one-character input to the data item TYPE-UPDATE.

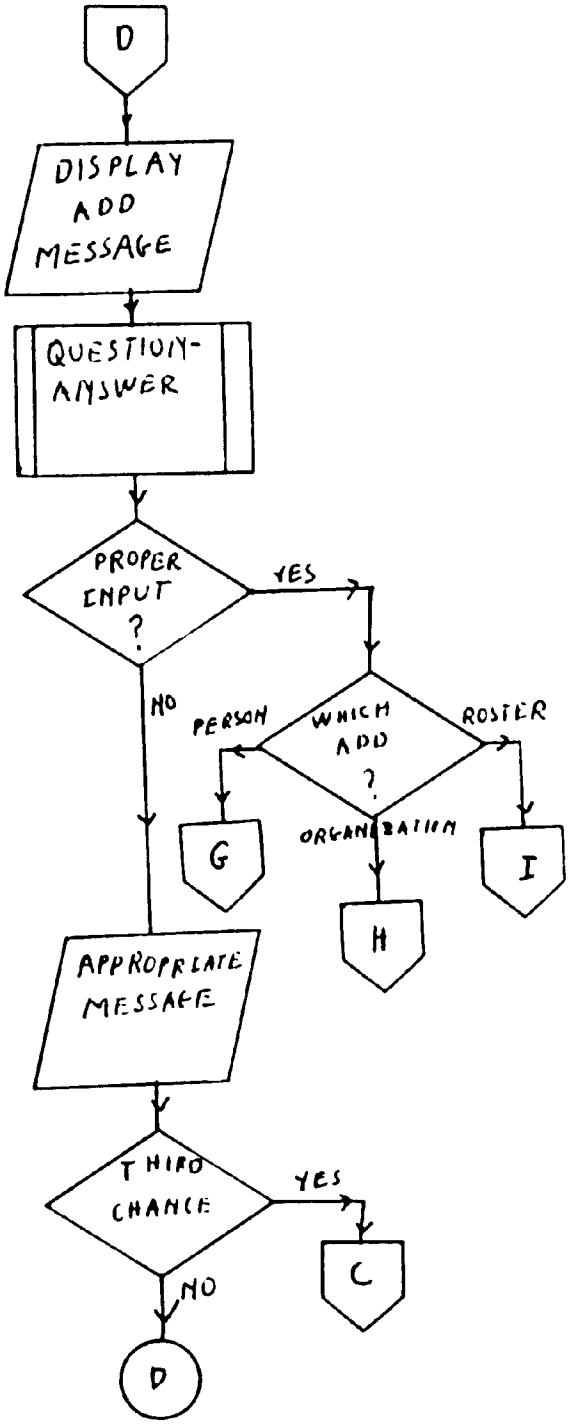
ROS-COM-ADD-OUTPUT (line 37300):

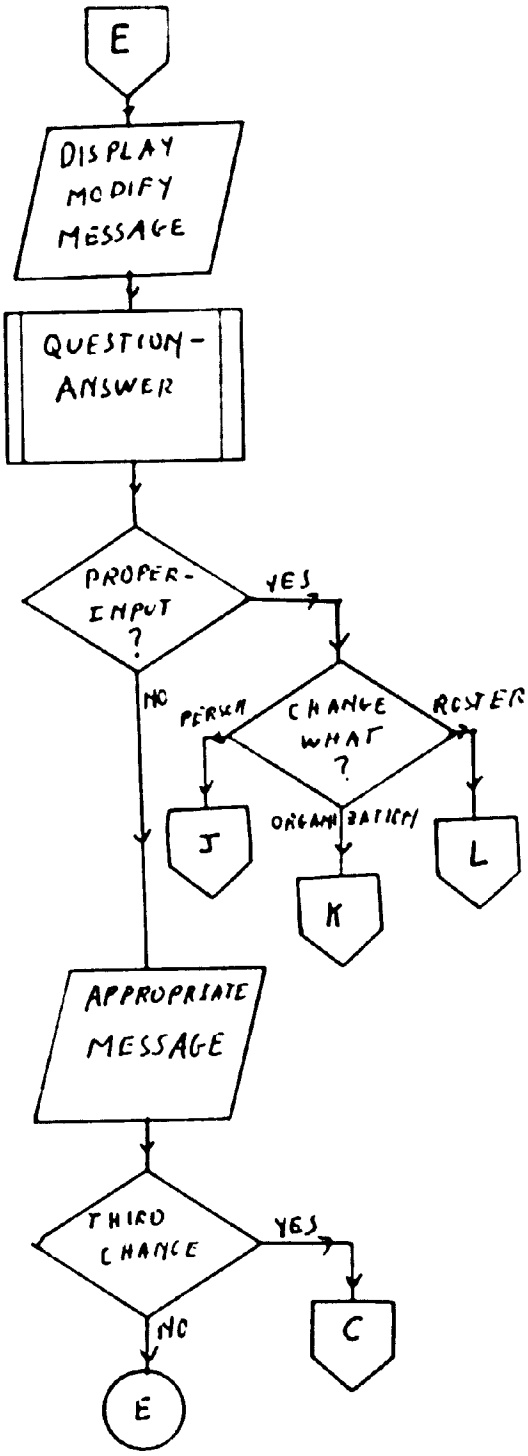
This routine writes the output report when a new roster, group, or committee is added to the data base.

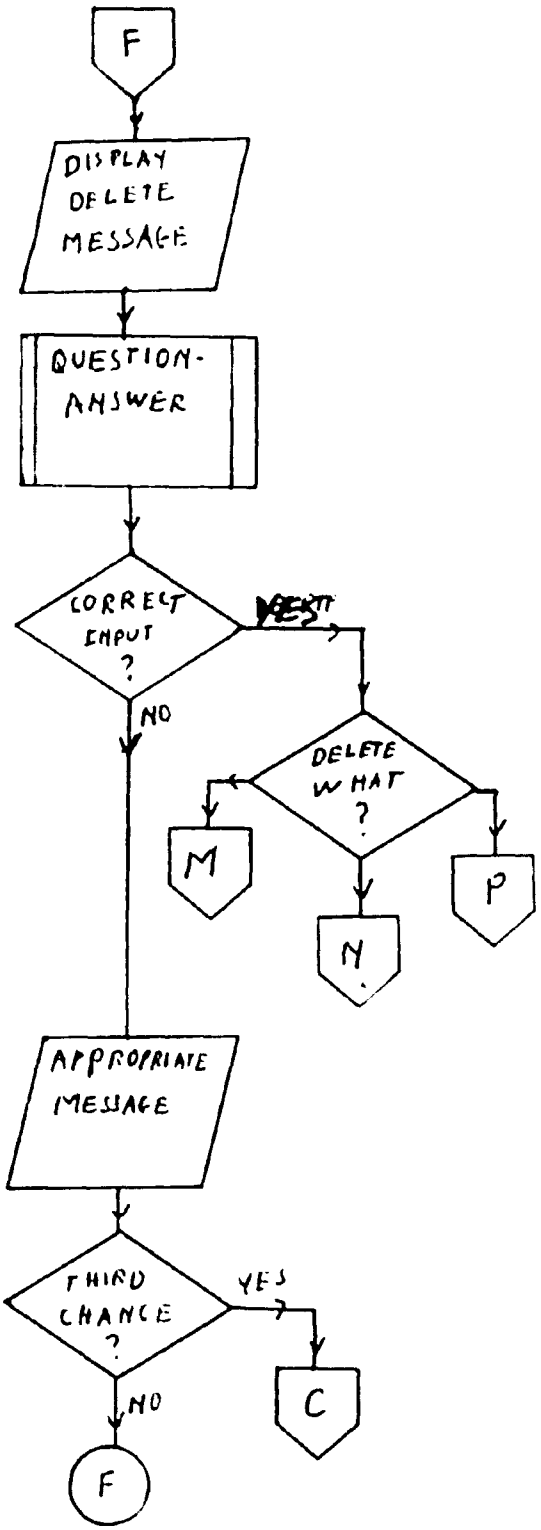
STORE-NEW-GROUP THRU SNG-EXIT (line 37700):

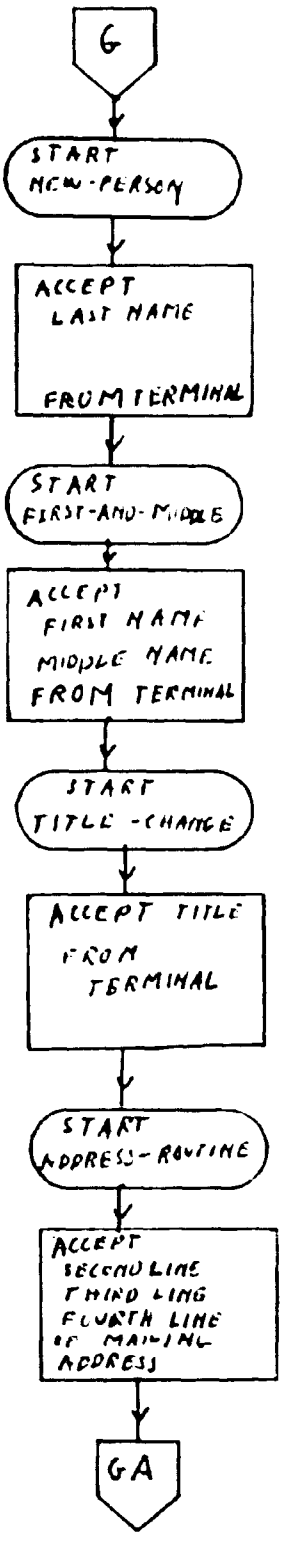
This routine is used when a new group is added to the data base. The routine asks for and accepts the roster number of the roster that the new group is contained in. Then the routine tries to find the associated ROSTER-REC record. If the find attempt is successful, the value 4 is moved to KOUNT.

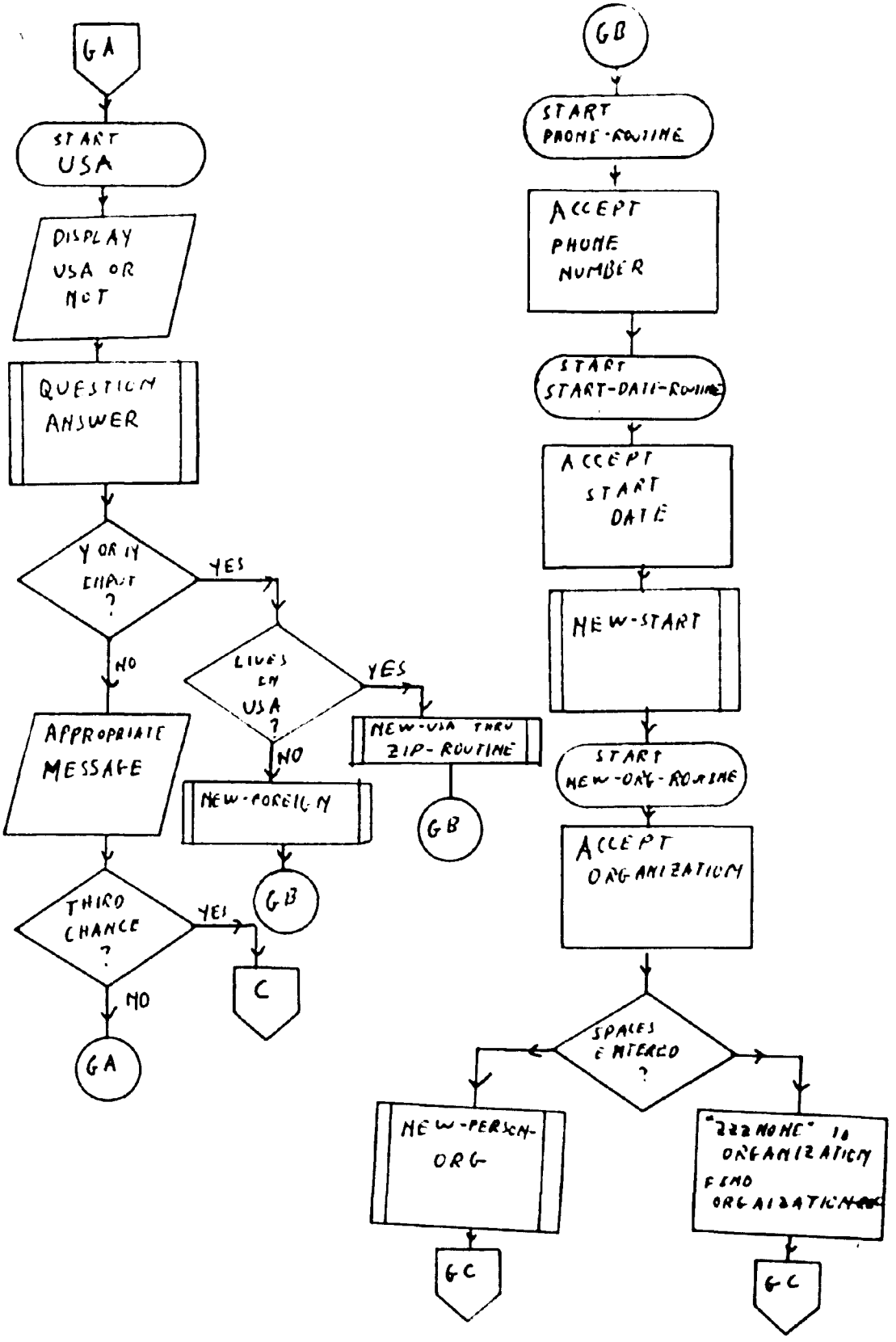

START UPDATE
OPEN OUTPUT FILES
OPEN DATABASE AREAS
HEADER-ROUTINE
A
USER OPTIONS MESSAGE
QUESTION-ANSWER
PROPER INPUT ?
YES
B
NO
APPROPRIATE MESSAGE
THIRD CHANCE ?
C
NO
A
B
TYPE-UPDATE ?
ADD NEW RECORD
END PROGRAM
D
MODIFY
DELETE
C
E
F
C
CLOSE FILES
CLOSE DATABASE AREAS
STOP RUN

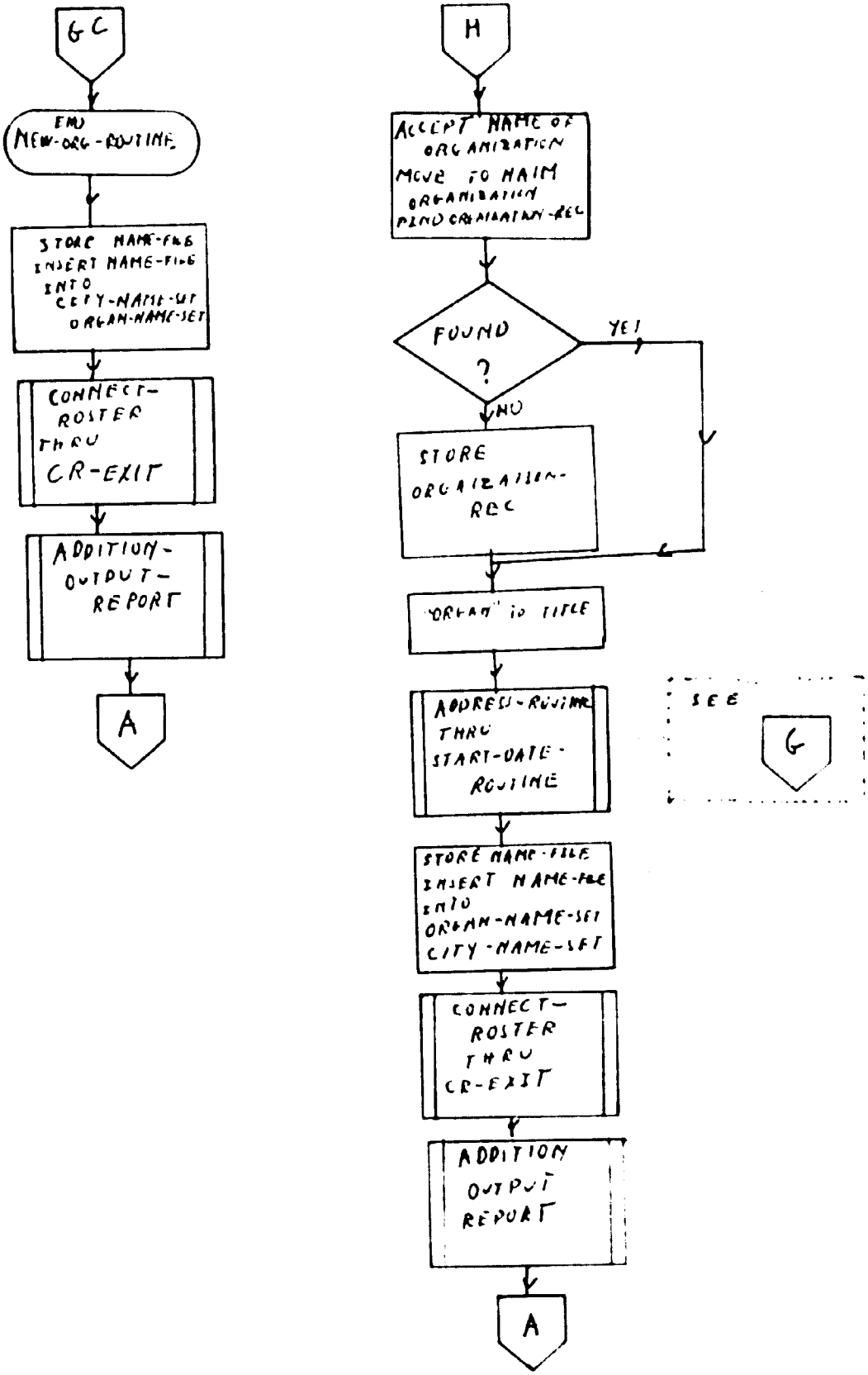

D
DISPLAY ADD MESSAGE
QUESTION-ANSWER
PROPER INPUT ?
YES
NO
WHICH ADD ?
PERSON
ROSTER
ORGANIZATION
G
H
I
APPROPRIATE MESSAGE
THIRD CHANCE
YES
NO
C
D

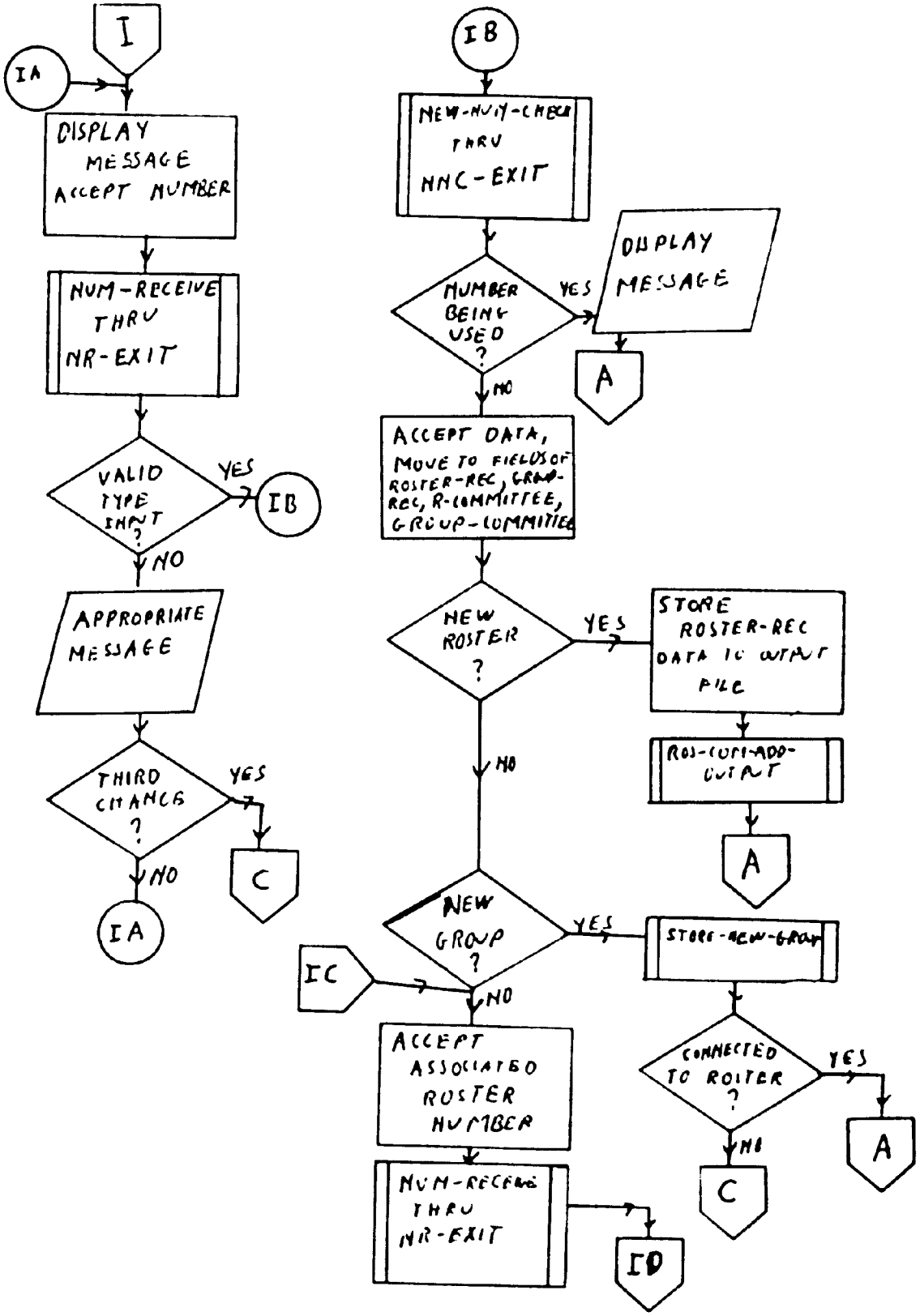

E
DISPLAY MODIFY MESSAGE
QUESTION-ANSWER
PROPER-INPUT ?
YES
NO
CHANGE WHAT ?
PERSON
ROSTER
ORGANIZATION
J
K
L
APPROPRIATE MESSAGE
THIRD CHANCE
YES
NO
C
E

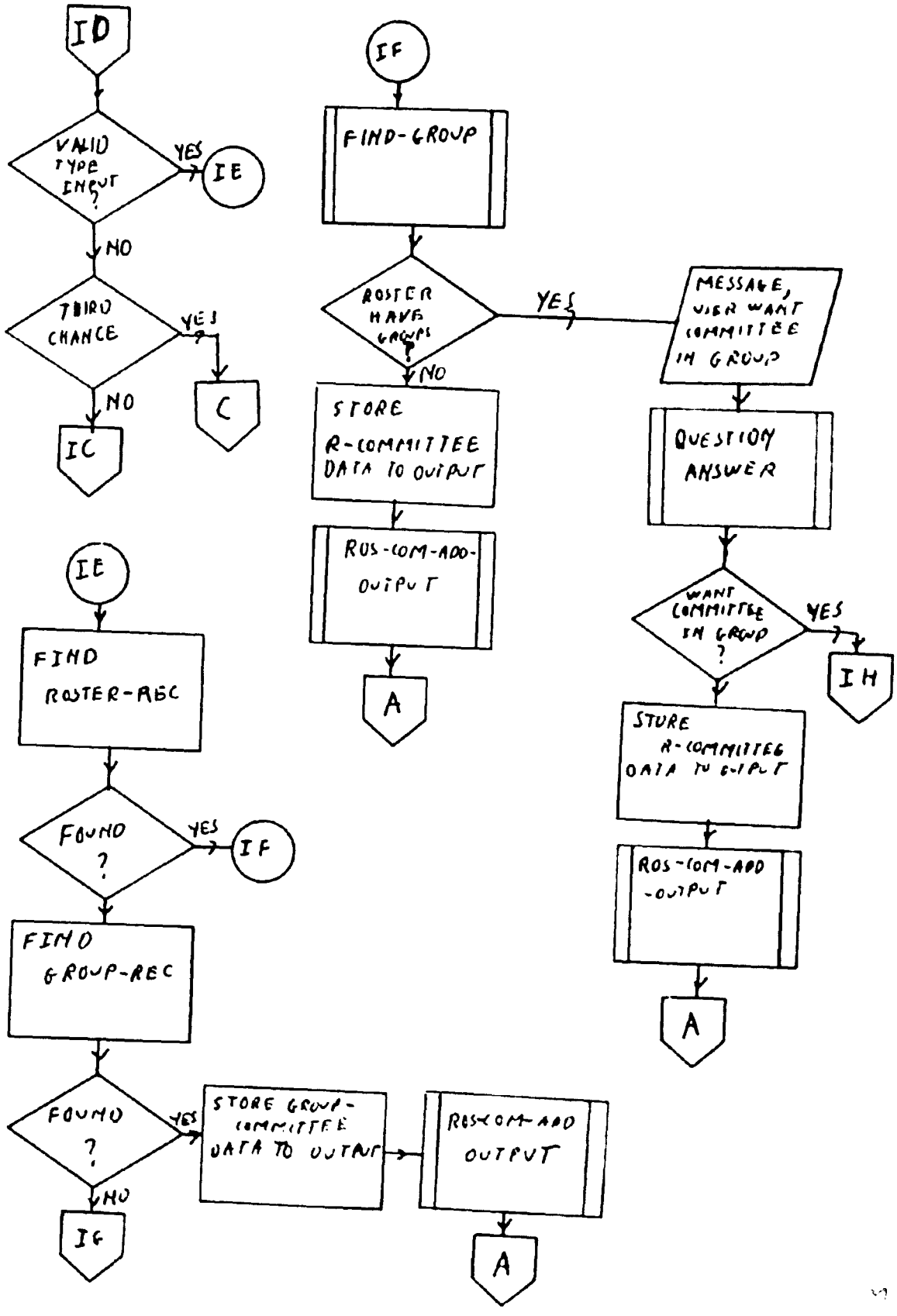

F
DISPLAY DELETE MESSAGE
QUESTION-ANSWER
CORRECT INPUT ?
YES
NO
DELETE WHAT ?
M
N
P
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
NO
C
F

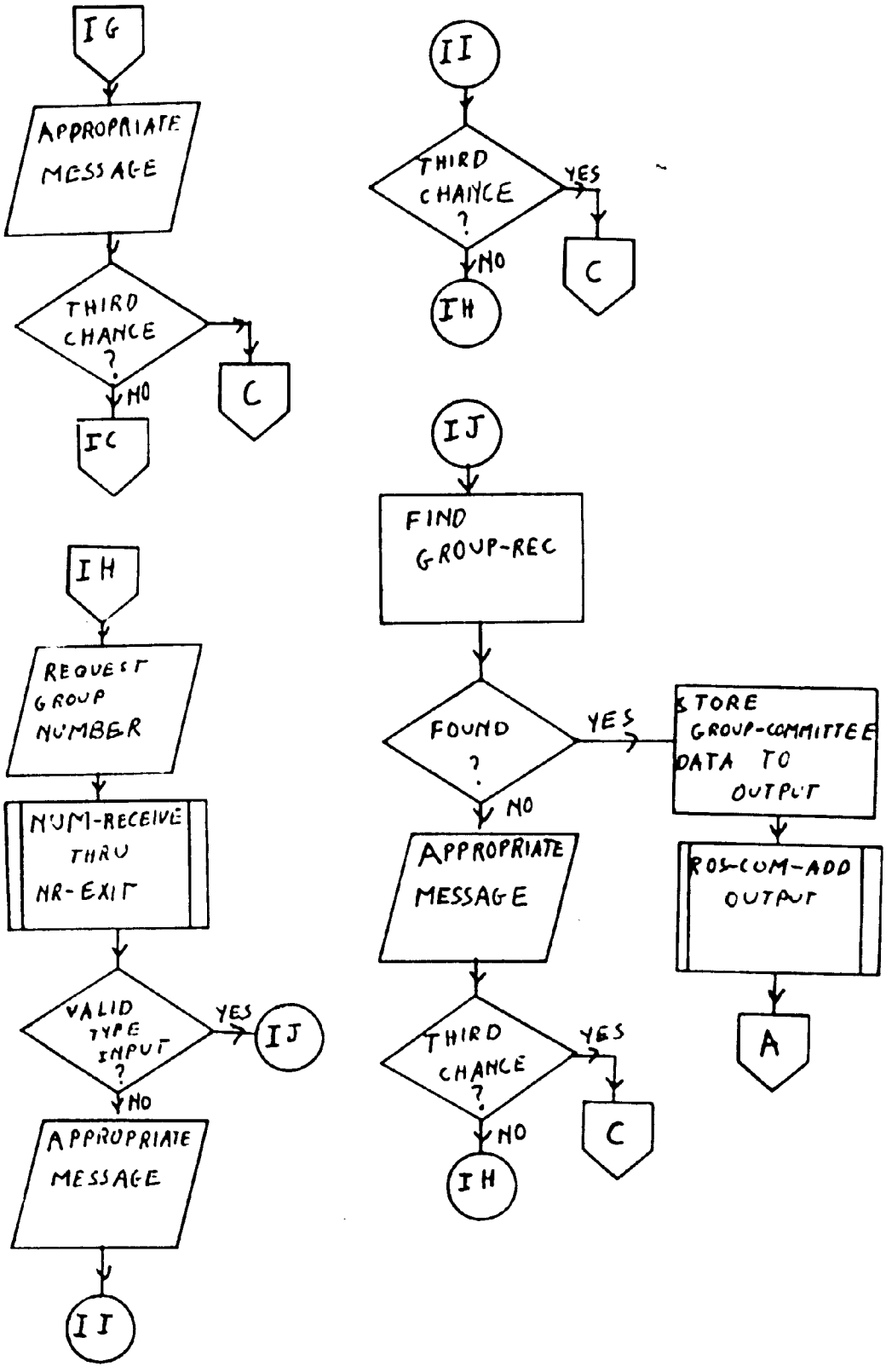

G
START NEW-PERSON
ACCEPT LAST NAME FROM TERMINAL
START FIRST-AND-MIDDLE
ACCEPT FIRST NAME MIDDLE NAME FROM TERMINAL
START TITLE-CHANGE
ACCEPT TITLE FROM TERMINAL
START ADDRESS-ROUTINE
ACCEPT SECOND LINE THIRD LINE FOURTH LINE OF MAILING ADDRESS
GA

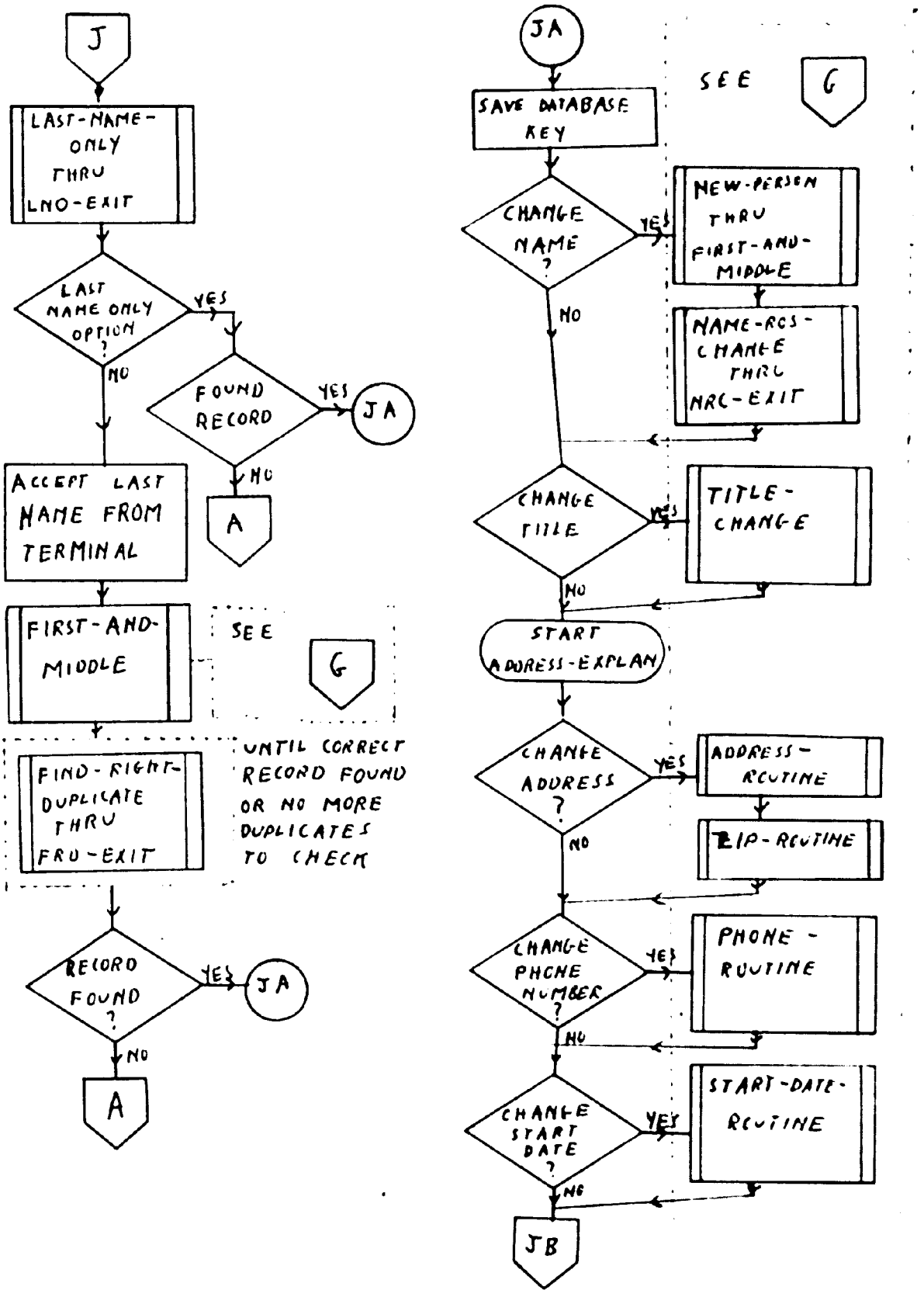

GA
START USA
DISPLAY USA OR NOT
QUESTION ANSWER
Y OR N EMPTY ?
YES
NO
LIVES IN USA ?
YES
NO
NEW-USA THRU ZIP-ROUTINE
GB
NEW-FOREIGN
GB
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
NO
GB
C
GA
GB
START PHONE-ROUTINE
ACCEPT PHONE NUMBER
START START-DATE-ROUTINE
ACCEPT START DATE
NEW-START
START NEW-ORG-ROUTINE
ACCEPT ORGANIZATION
SPACES ENTERED ?
NEW-PERSON-ORG
GC
"222 HOME" TO ORGANIZATION
FIND ORGANIZATION
GC

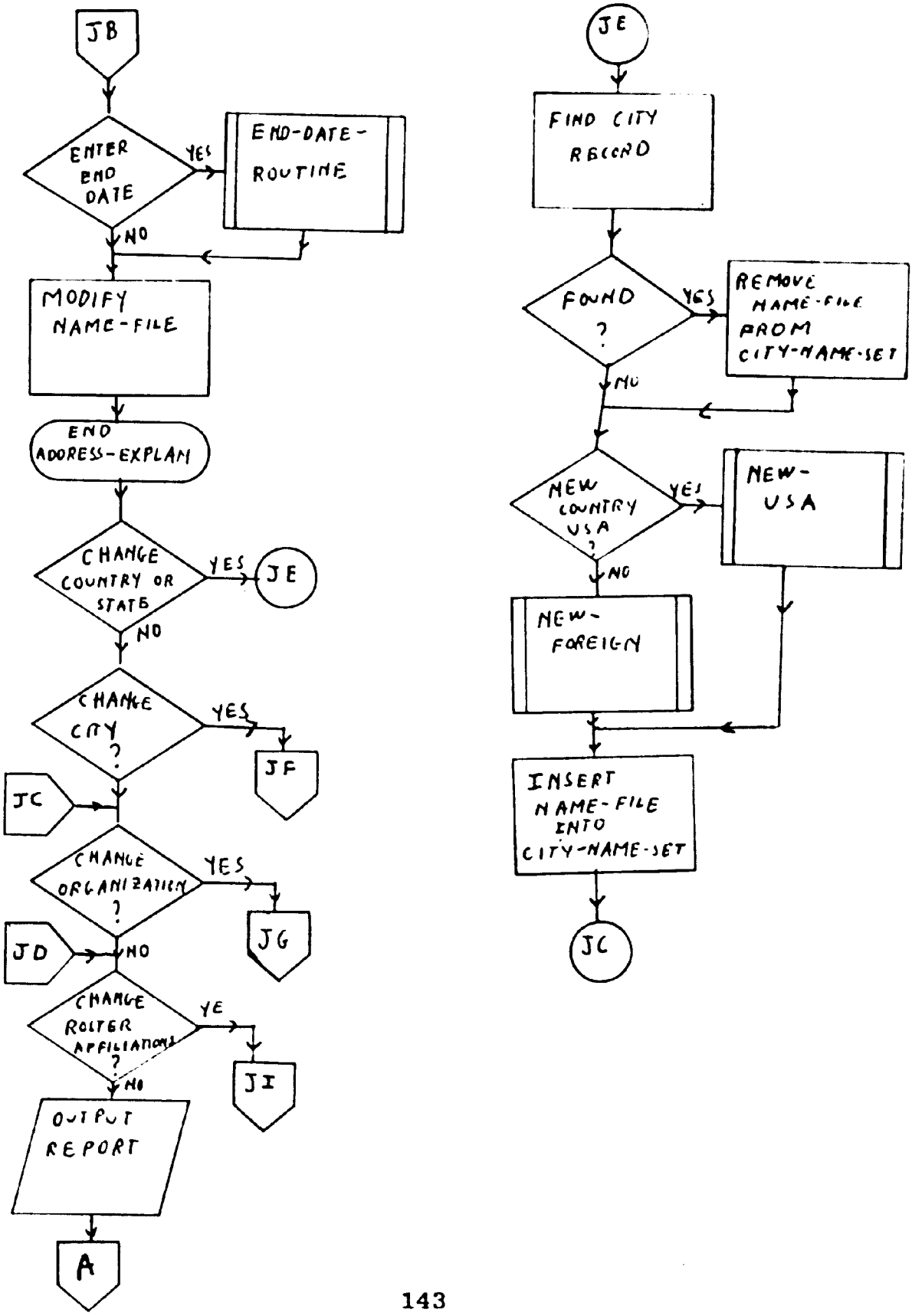

6C
END NEW-ORG-ROUTINE
STORE NAME-FILE
INSERT NAME-FILE INTO CITY-NAME-SET ORGAN-NAME-SET
CONNECT-ROSTER THRU CR-EXIT
ADDITION-OUTPUT-REPORT
A
H
ACCEPT NAME OF ORGANIZATION
MOVE TO MAIN ORGANIZATION
FIND ORGANIZATION-REC
FOUND ?
YES
NO
STORE ORGANIZATION-REC
"ORGAN" to TITLE
ADDRESS-ROUTINE THRU START-DATE-ROUTINE
SEE 6
STORE NAME-FILE
INSERT NAME-FILE INTO ORGAN-NAME-SET CITY-NAME-SET
CONNECT-ROSTER THRU CR-EXIT
ADDITION OUTPUT REPORT
A

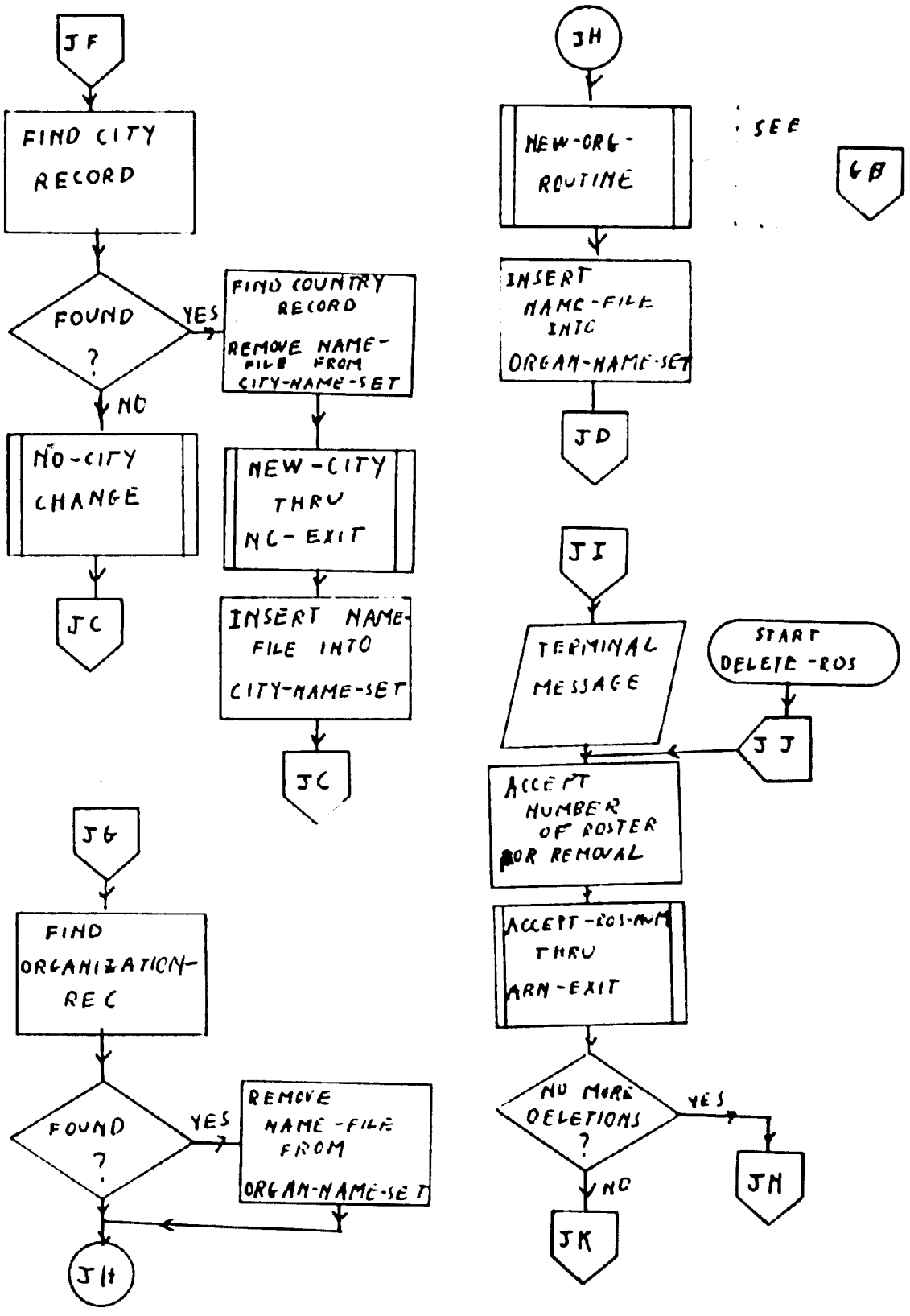

I
IA
DISPLAY MESSAGE ACCEPT NUMBER
NUM-RECEIVE THRU NR-EXIT
VALID TYPE INPUT ?
YES
IB
NO
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
C
NO
IA
IB
NEW-NUM-CHECK THRU NNC-EXIT
NUMBER BEING USED ?
YES
DISPLAY MESSAGE
A
NO
ACCEPT DATA, MOVE TO FIELDS OF ROSTER-REC, GROUP-REC, R-COMMITTEE, GROUP-COMMITTEE
NEW ROSTER ?
YES
STORE ROSTER-REC DATA TO OUTPUT FILE
ROS-CONT-ADD-OUTPUT
A
NO
NEW GROUP ?
YES
STORE-NEW-GROUP
IC
NO
ACCEPT ASSOCIATED ROSTER NUMBER
NUM-RECEIVE THRU NR-EXIT
ID
CONNECTED TO ROSTER ?
YES
A
NO
C

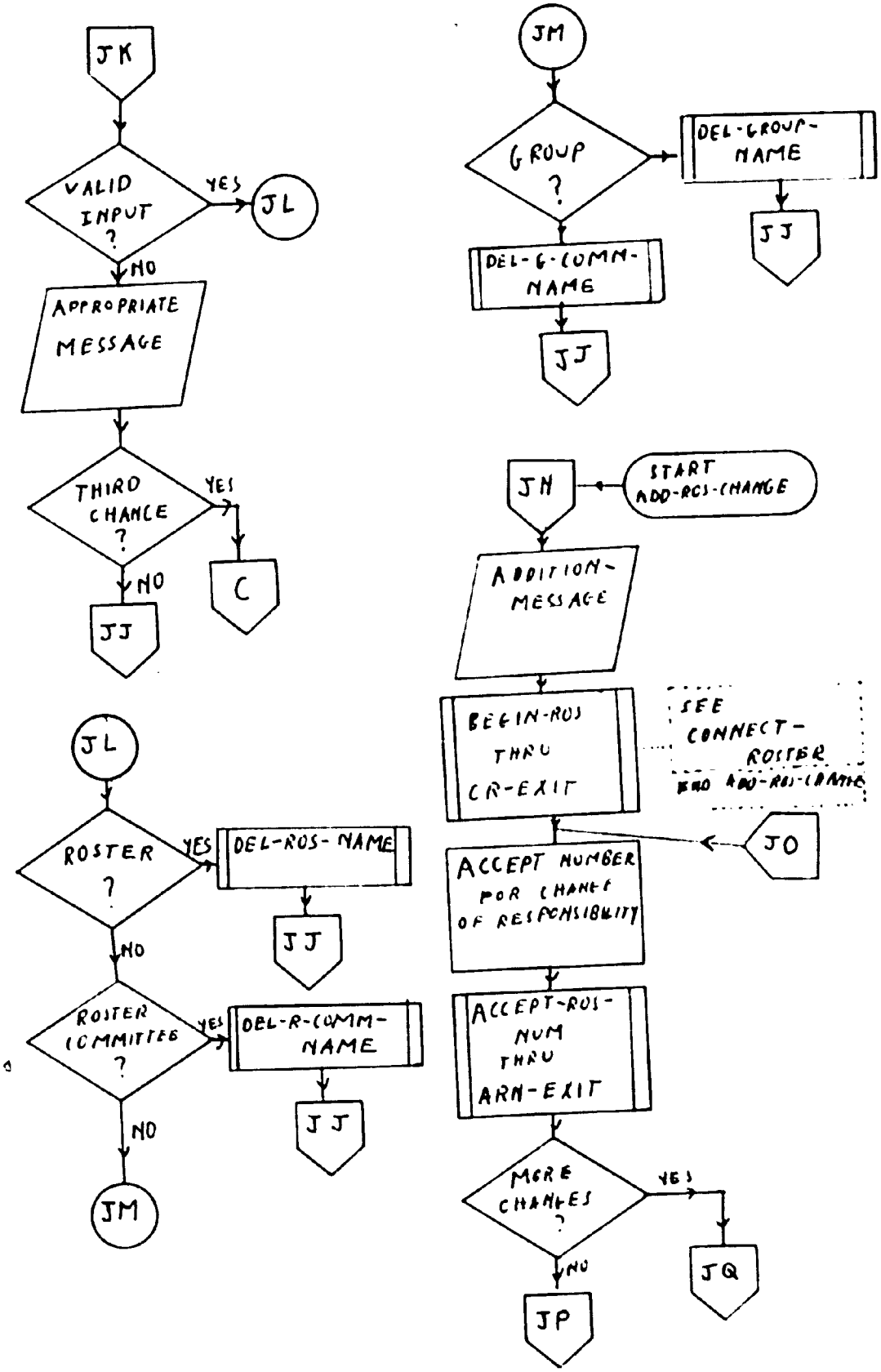

ID
VALID TYPE INPUT ?
YES
IE
NO
THIRD CHANCE
YES
C
NO
IC
IE
FIND ROSTER-REC
FOUND ?
YES
IF
NO
IG
FIND GROUP-REC
FOUND ?
YES
STORE GROUP-COMMITTEE DATA TO OUTPUT
ROS-COM-ADD OUTPUT
A
IF
FIND-GROUP
ROSTER HAVE GROUPS ?
YES
MESSAGE, USER WANT COMMITTEE IN GROUP
NO
STORE R-COMMITTEE DATA TO OUTPUT
ROS-COM-ADD-OUTPUT
A
QUESTION ANSWER
WANT COMMITTEE IN GROUP ?
YES
IH
STORE R-COMMITTEE DATA TO OUTPUT
ROS-COM-ADD -OUTPUT
A

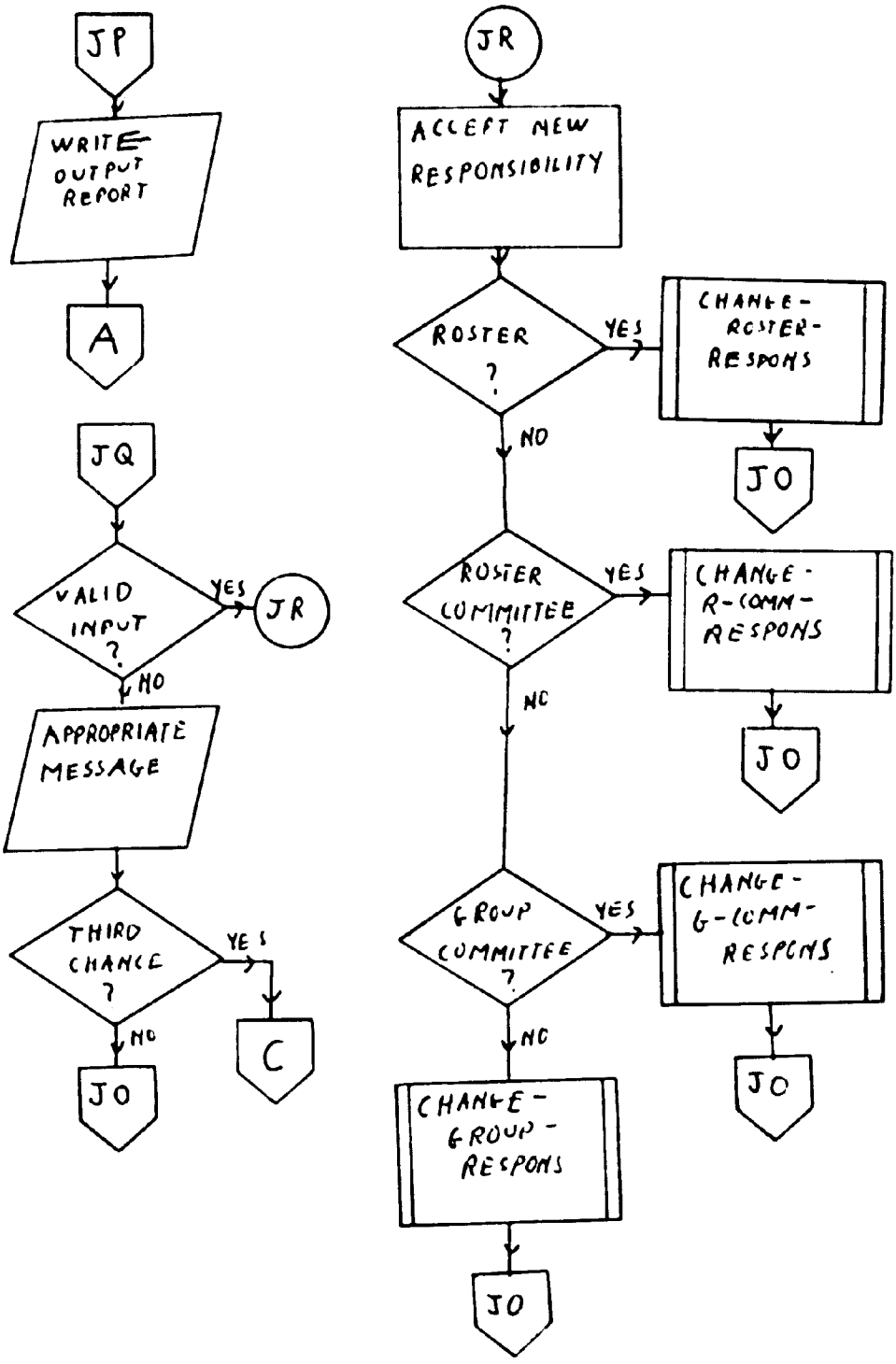

IG
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
NO
C
IC
II
THIRD CHANCE ?
YES
NO
C
IH
IH
REQUEST GROUP NUMBER
NUM-RECEIVE THRU NR-EXIT
VALID TYPE INPUT ?
YES
IJ
NO
APPROPRIATE MESSAGE
II
IJ
FIND GROUP-REC
FOUND ?
YES
STORE GROUP-COMMITTEE DATA TO OUTPUT
NO
APPROPRIATE MESSAGE
ROS-COM-ADD OUTPUT
A
THIRD CHANCE ?
YES
C
NO
IH

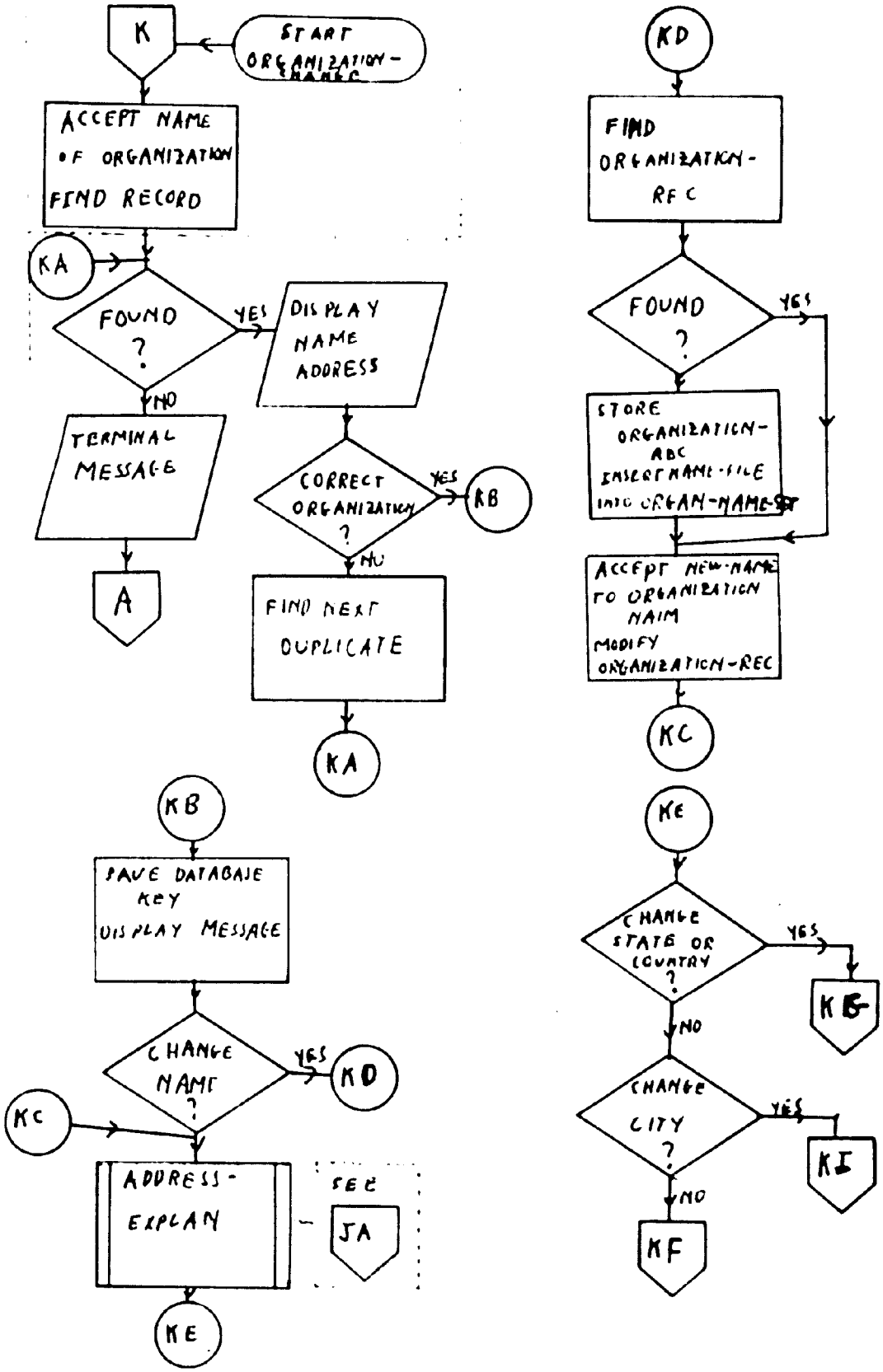

J
LAST-NAME-ONLY THRU LNO-EXIT
LAST NAME ONLY OPTION ?
YES
NO
FOUND RECORD
YES
JA
NO
A
ACCEPT LAST NAME FROM TERMINAL
FIRST-AND-MIDDLE
SEE G
UNTIL CORRECT RECORD FOUND OR NO MORE DUPLICATES TO CHECK
FIND-RIGHT-DUPLICATE THRU FRD-EXIT
RECORD FOUND ?
YES
JA
NO
A
JA
SAVE DATABASE KEY
SEE G
CHANGE NAME ?
YES
NEW-PERSON THRU FIRST-AND-MIDDLE
NAME-RCS-CHANGE THRU NRC-EXIT
NO
CHANGE TITLE
YES
TITLE-CHANGE
NO
START ADDRESS-EXPLAN
CHANGE ADDRESS ?
YES
ADDRESS-ROUTINE
ZIP-ROUTINE
NO
CHANGE PHONE NUMBER ?
YES
PHONE-ROUTINE
NO
CHANGE START DATE ?
YES
START-DATE-ROUTINE
NO
JB

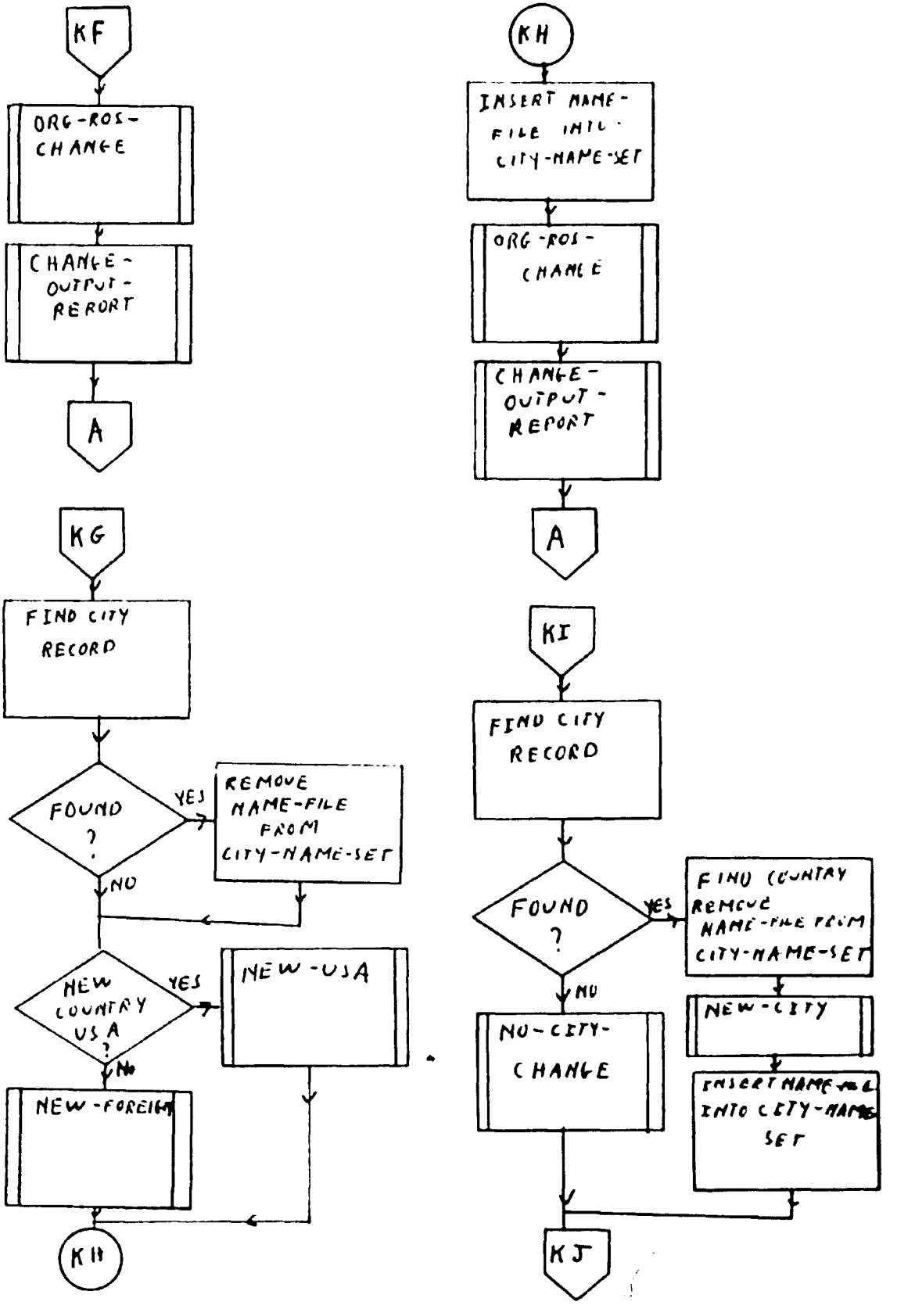

JB
ENTER END DATE
YES
END-DATE-ROUTINE
NO
MODIFY NAME-FILE
END ADDRESS-EXPLAN
CHANGE COUNTRY OR STATE
YES
JE
NO
CHANGE CITY ?
YES
JF
JC
CHANGE ORGANIZATION ?
YES
JG
JD
NO
CHANGE ROUTER AFFILIATION ?
YE
JI
NO
OUTPUT REPORT
A
JE
FIND CITY RECORD
FOUND ?
YES
REMOVE NAME-FILE FROM CITY-NAME-SET
NO
NEW COUNTRY USA ?
YES
NEW-USA
NO
NEW-FOREIGN
INSERT NAME-FILE INTO CITY-NAME-SET
JC

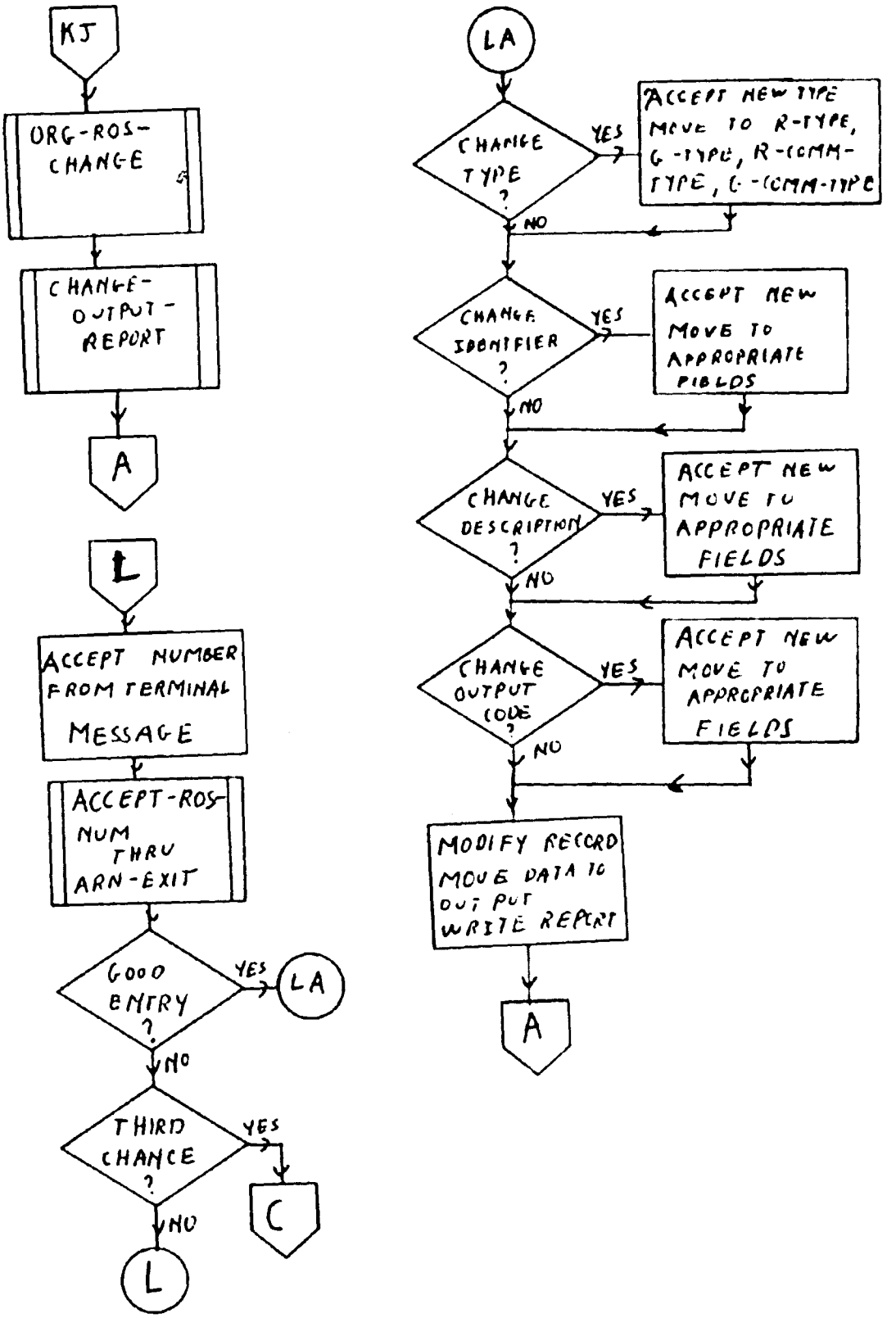

JF
FIND CITY RECORD
FOUND ?
YES
NO
FIND COUNTRY RECORD
REMOVE NAME-FILE FROM CITY-NAME-SET
NO-CITY CHANGE
NEW-CITY THRU NC-EXIT
JC
INSERT NAME-FILE INTO CITY-NAME-SET
JC
JG
FIND ORGANIZATION-REC
FOUND ?
YES
REMOVE NAME-FILE FROM ORGAN-NAME-SET
JH
NEW-ORG-ROUTINE
SEE 6B
INSERT NAME-FILE INTO ORGAN-NAME-SET
JD
JI
TERMINAL MESSAGE
START DELETE-ROS
JJ
ACCEPT NUMBER OF ROSTER FOR REMOVAL
ACCEPT-ROS-NUM THRU ARM-EXIT
NO MORE DELETIONS ?
YES
NO
JM
JK

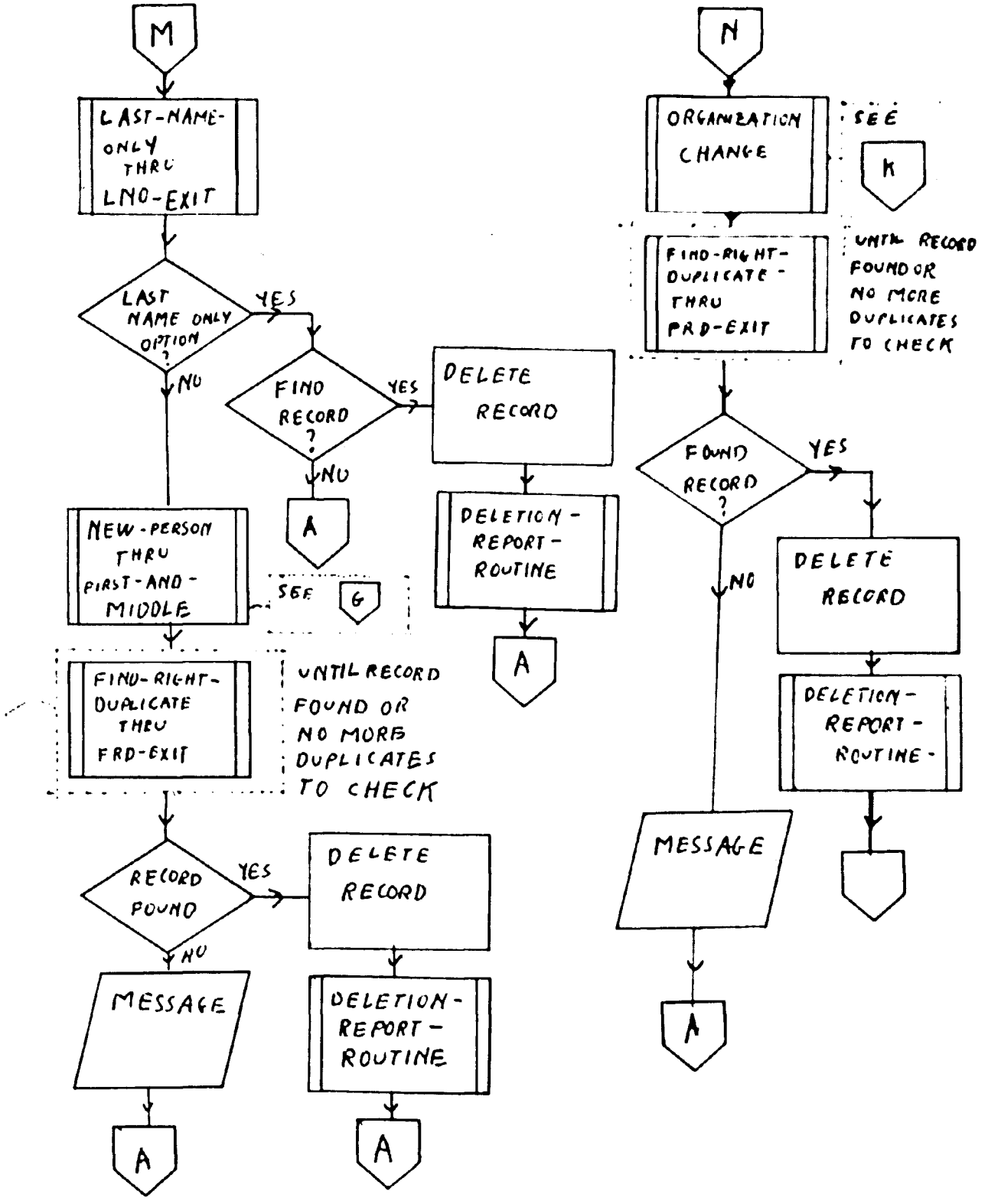

JK
VALID INPUT ?
YES
JL
NO
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
C
NO
JJ
JM
GROUP ?
DEL-GROUP-NAME
JJ
DEL-G-COMM-NAME
JJ
JN
START ADD-ROS-CHANGE
ADDITION-MESSAGE
BEGIN-ROS THRU CR-EXIT
SEE CONNECT-ROSTER AND ADD-ROS-CHANGE
JO
ACCEPT NUMBER FOR CHANGE OF RESPONSIBILITY
ACCEPT-ROS-NUM THRU ARN-EXIT
MORE CHANGES ?
YES
JQ
NO
JP
JL
ROSTER ?
YES
DEL-ROS-NAME
JJ
NO
ROSTER COMMITTEE ?
YES
DEL-R-COMM-NAME
JJ
NO
JM

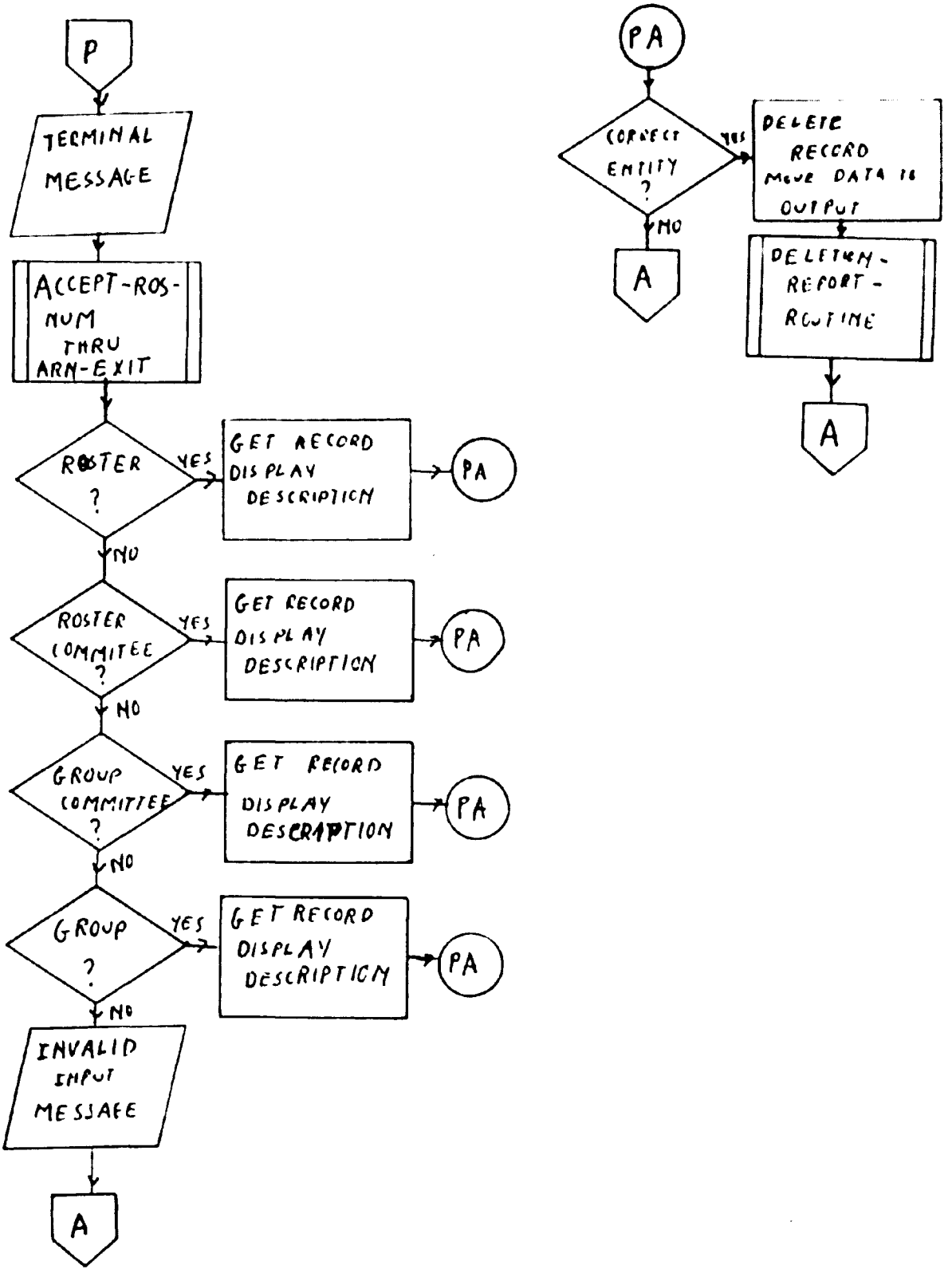

JP
WRITE OUTPUT REPORT
A
JQ
VALID INPUT ?
YES
JR
NO
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
C
NO
JO
JR
ACCEPT NEW RESPONSIBILITY
ROSTER ?
YES
CHANGE-ROSTER-RESPONS
NO
JO
ROSTER COMMITTEE ?
YES
CHANGE-R-COMM-RESPONS
NO
JO
GROUP COMMITTEE ?
YES
CHANGE-G-COMM-RESPONS
NO
JO
CHANGE-GROUP-RESPONS
JO

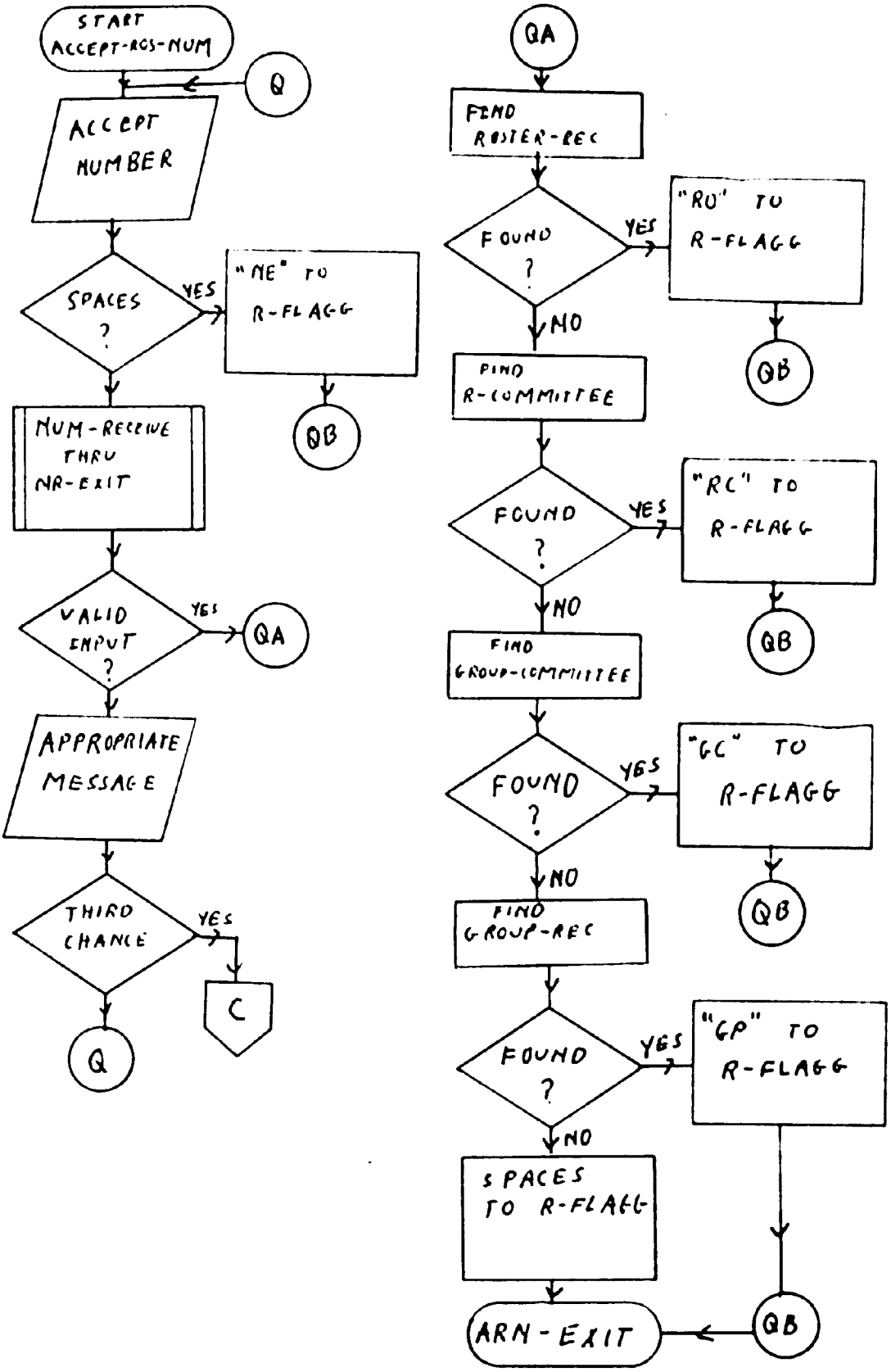

K
START ORGANIZATION-NAME-C
ACCEPT NAME OF ORGANIZATION FIND RECORD
KA
FOUND ?
YES
DISPLAY NAME ADDRESS
NO
TERMINAL MESSAGE
A
CORRECT ORGANIZATION ?
YES
KB
NO
FIND NEXT DUPLICATE
KA
KB
SAVE DATABASE KEY
DISPLAY MESSAGE
CHANGE NAME ?
YES
KD
KC
ADDRESS-EXPLAN
SEE
JA
KE
KD
FIND ORGANIZATION-REC
FOUND ?
YES
STORE ORGANIZATION-ABC INSERT NAME-FILE INTO ORGAN-NAME-ST
ACCEPT NEW-NAME TO ORGANIZATION NAME
MODIFY ORGANIZATION-REC
KC
KE
CHANGE STATE OR COUNTRY ?
YES
KG
NO
CHANGE CITY ?
YES
KI
NO
KF

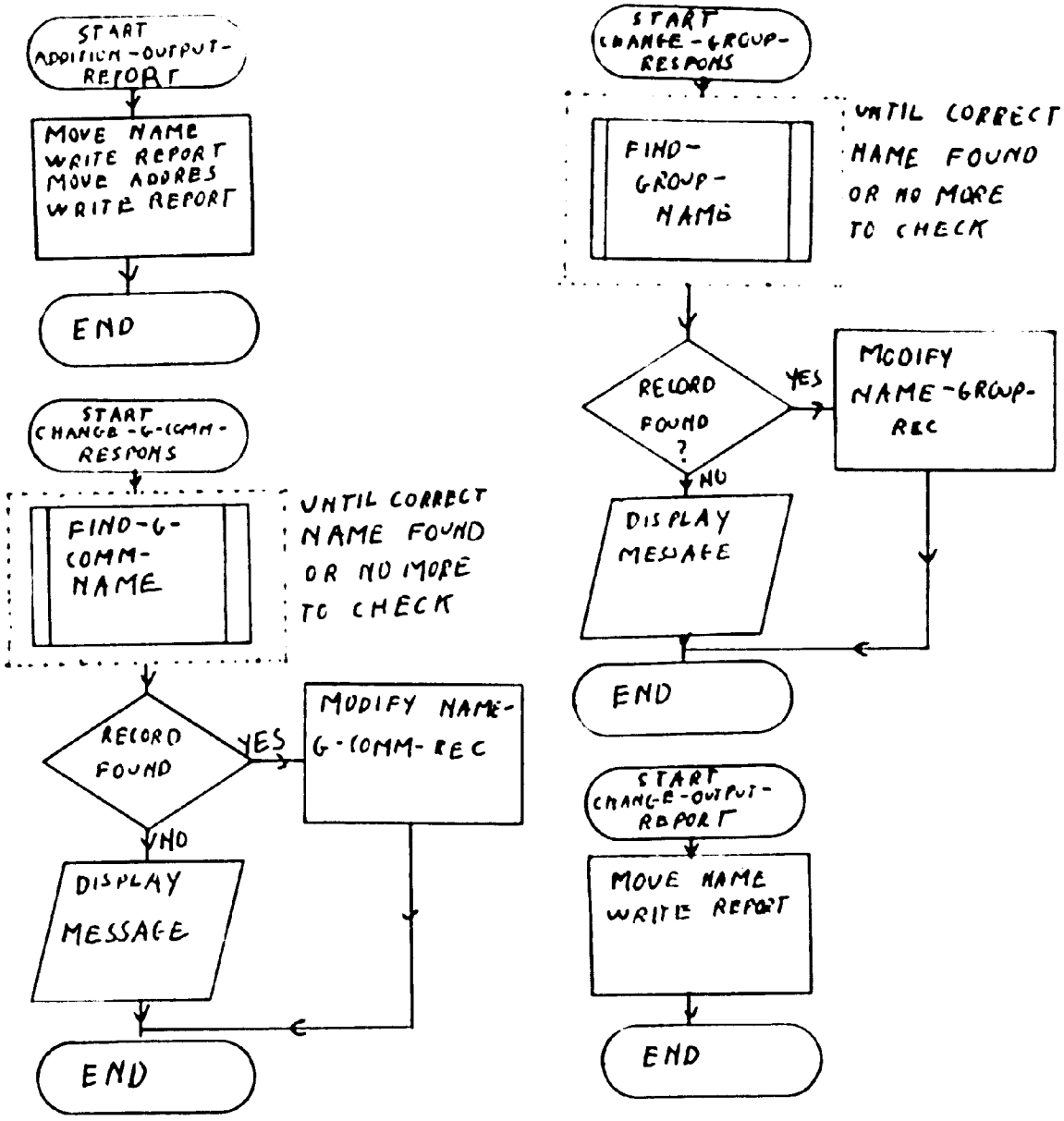

KF
ORG-ROS-CHANGE
CHANGE-OUTPUT-REPORT
A
KG
FIND CITY RECORD
FOUND ?
YES
REMOVE NAME-FILE FROM CITY-NAME-SET
NO
NEW COUNTRY USA ?
YES
NEW-USA
NO
NEW-FOREIGN
KH
KH
INSERT NAME-FILE INTO CITY-NAME-SET
ORG-ROS-CHANGE
CHANGE-OUTPUT-REPORT
A
KI
FIND CITY RECORD
FOUND ?
YES
FIND COUNTRY
REMOVE NAME-FILE FROM CITY-NAME-SET
NEW-CITY
INSERT NAME-FILE INTO CITY-NAME-SET
NO
NO-CITY-CHANGE
KJ

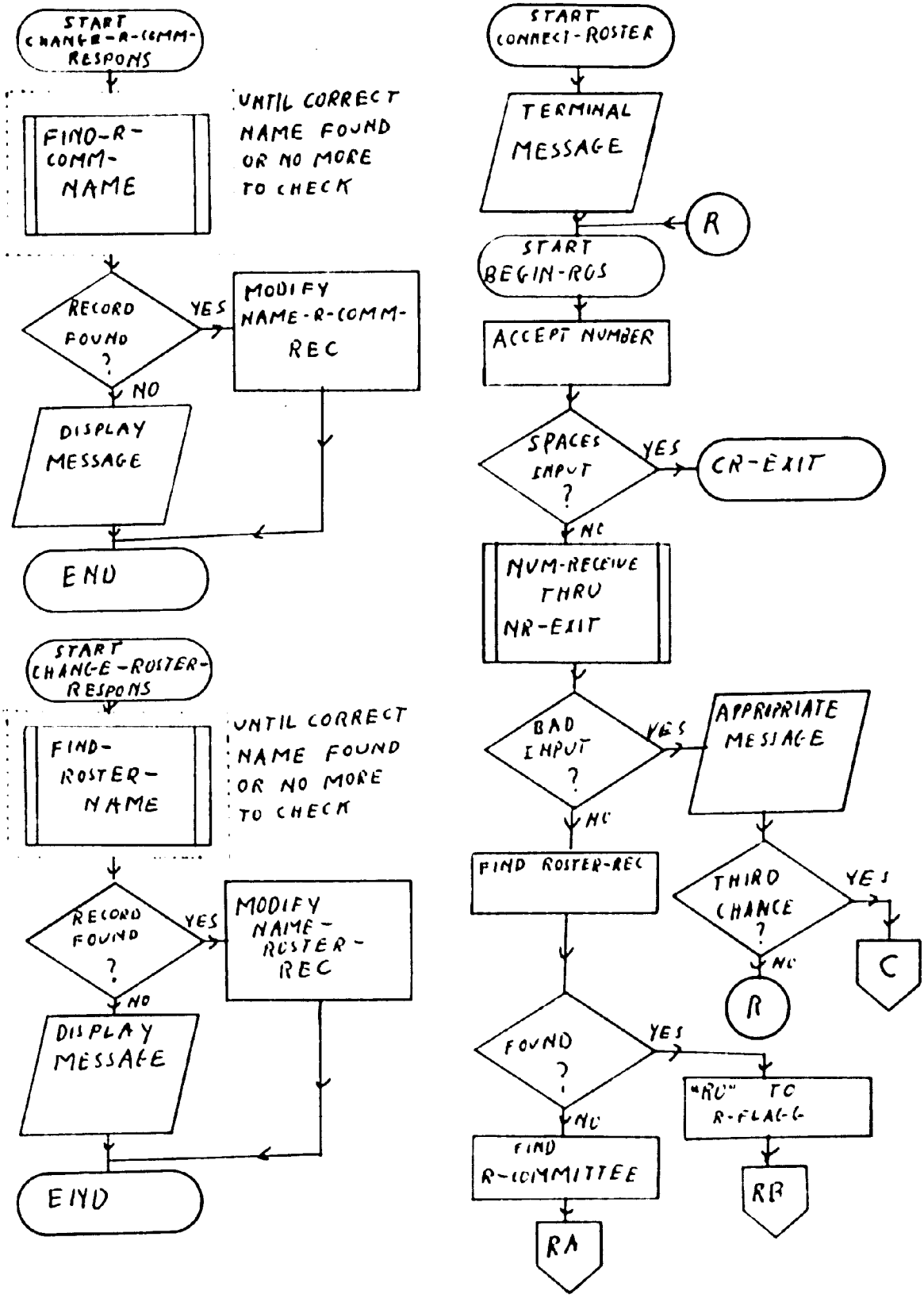

KJ
ORG-ROS-CHANGE
CHANGE-OUTPUT-REPORT
A
L
ACCEPT NUMBER FROM TERMINAL MESSAGE
ACCEPT-ROS-NUM THRU ARN-EXIT
GOOD ENTRY ?
YES
LA
NO
THIRD CHANCE ?
YES
C
NO
L
LA
CHANGE TYPE ?
YES
ACCEPT NEW TYPE MOVE TO R-TYPE, G-TYPE, R-COMM-TYPE, G-COMM-TYPE
NO
CHANGE IDENTIFIER ?
YES
ACCEPT NEW MOVE TO APPROPRIATE FIELDS
NO
CHANGE DESCRIPTION ?
YES
ACCEPT NEW MOVE TO APPROPRIATE FIELDS
NO
CHANGE OUTPUT CODE ?
YES
ACCEPT NEW MOVE TO APPROPRIATE FIELDS
NO
MODIFY RECORD MOVE DATA TO OUTPUT WRITE REPORT
A

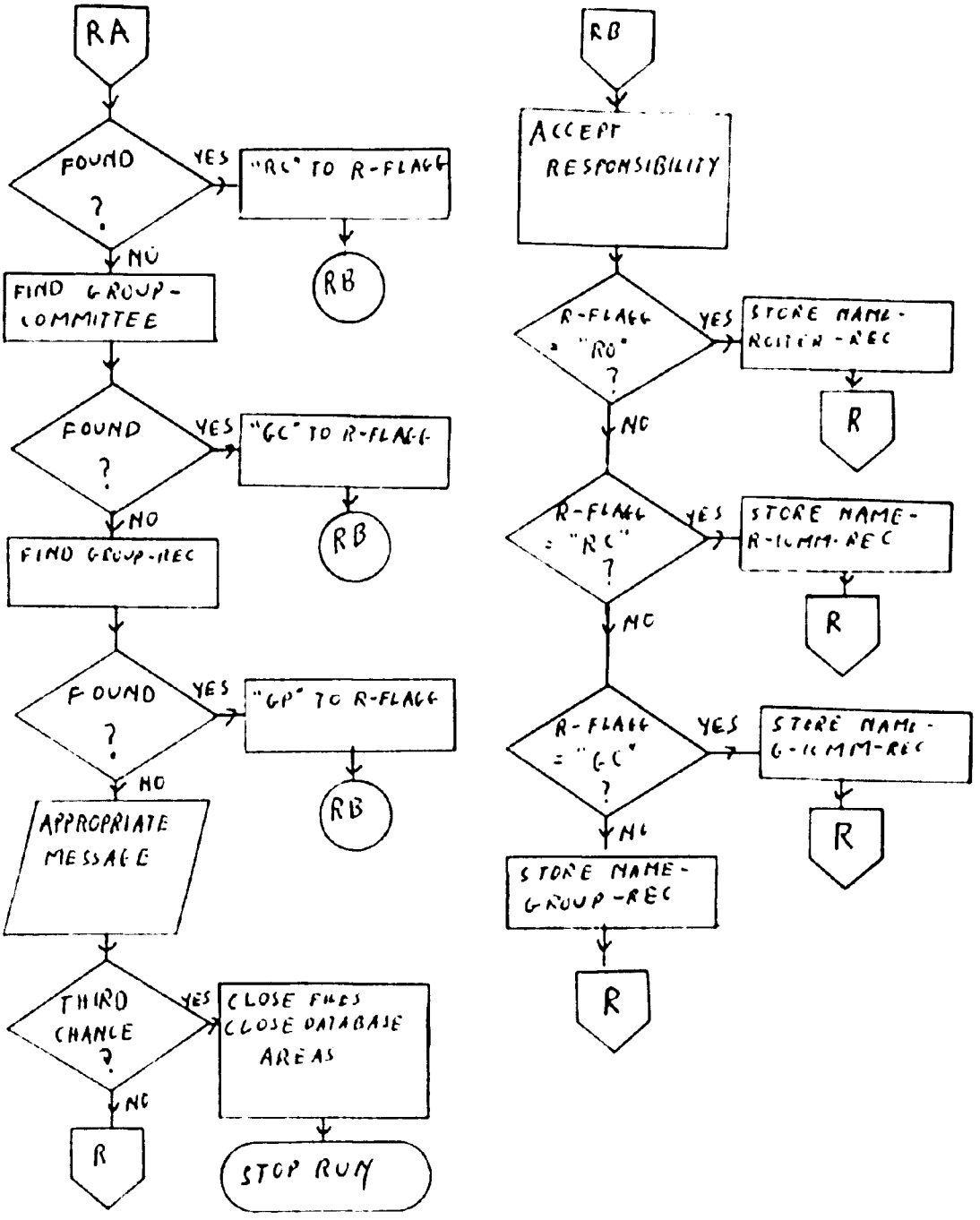

M
LAST-NAME-ONLY THRU LNO-EXIT
LAST NAME ONLY OPTION ?
YES
NO
FIND RECORD ?
YES
DELETE RECORD
DELETION-REPORT-ROUTINE
A
NO
A
NEW-PERSON THRU FIRST-AND-MIDDLE
SEE G
FIND-RIGHT-DUPLICATE THRU FRD-EXIT
UNTIL RECORD FOUND OR NO MORE DUPLICATES TO CHECK
RECORD FOUND
YES
DELETE RECORD
NO
MESSAGE
DELETION-REPORT-ROUTINE
A
A
N
ORGANIZATION CHANGE
SEE K
FIND-RIGHT-DUPLICATE-THRU PRD-EXIT
UNTIL RECORD FOUND OR NO MORE DUPLICATES TO CHECK
FOUND RECORD ?
YES
DELETE RECORD
DELETION-REPORT-ROUTINE
NO
MESSAGE
A

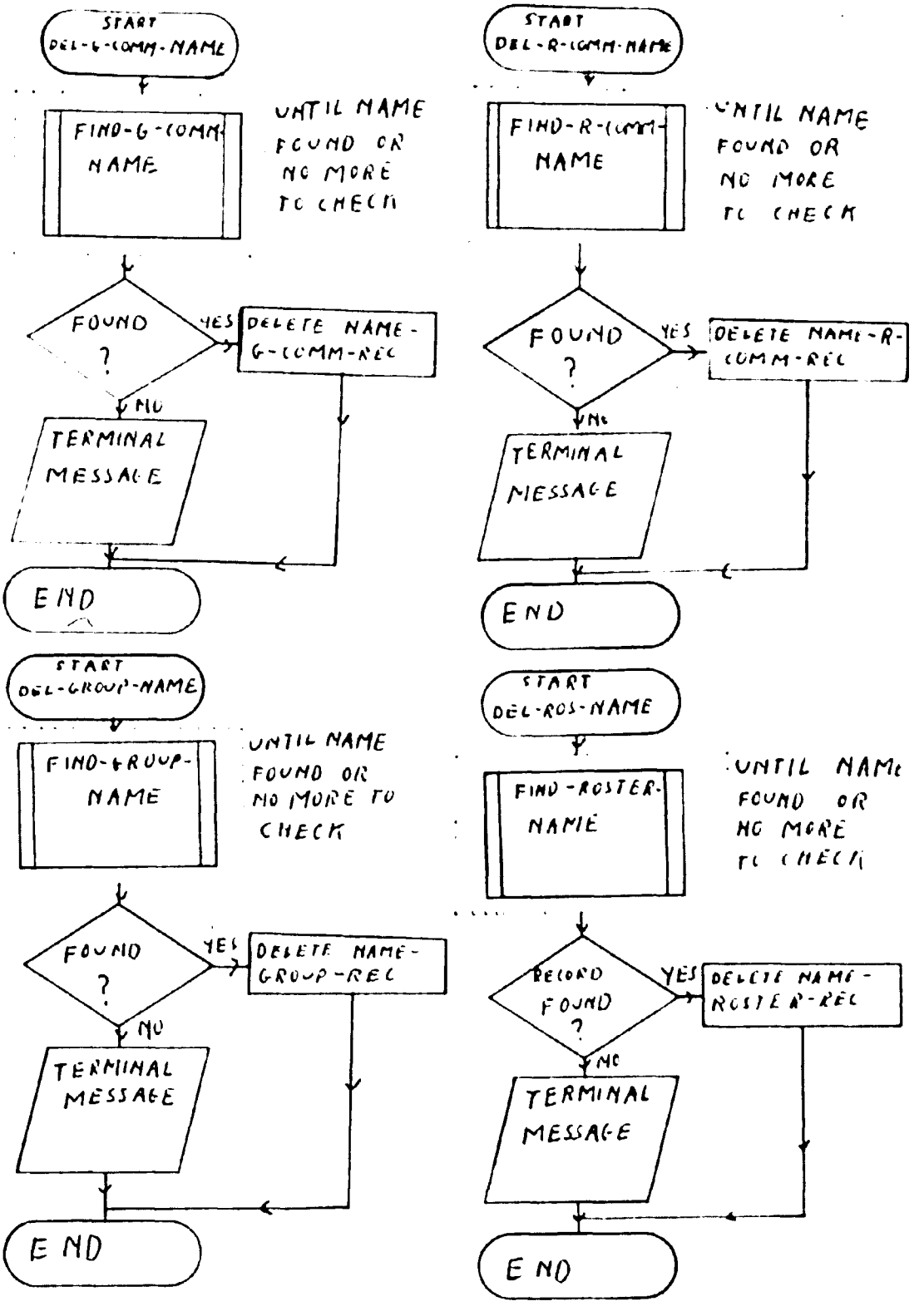

P
TERMINAL MESSAGE
ACCEPT-ROS-NUM THRU ARN-EXIT
ROSTER ?
YES
GET RECORD DISPLAY DESCRIPTION
PA
NO
ROSTER COMMITEE ?
YES
GET RECORD DISPLAY DESCRIPTION
PA
NO
GROUP COMMITTEE ?
YES
GET RECORD DISPLAY DESCRIPTION
PA
NO
GROUP ?
YES
GET RECORD DISPLAY DESCRIPTION
PA
NO
INVALID INPUT MESSAGE
A
PA
CORRECT ENTITY ?
YES
DELETE RECORD MOVE DATA TO OUTPUT
DELETION-REPORT-ROUTINE
A
NO
A

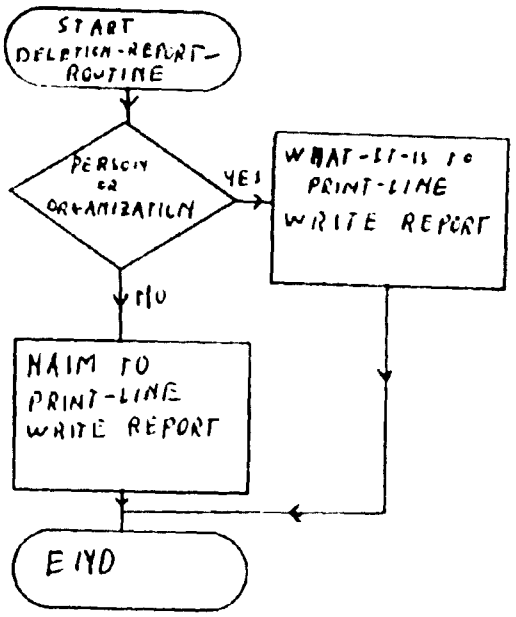

START
ACCEPT-ROS-NUM
Q
ACCEPT
NUMBER
SPACES
?
YES
"NE" TO
R-FLAGG
QB
NUM-RECEIVE
THRU
NR-EXIT
VALID
INPUT
?
YES
QA
APPROPRIATE
MESSAGE
THIRD
CHANCE
YES
C
Q
QA
FIND
ROSTER-REC
FOUND
?
YES
"RO" TO
R-FLAGG
QB
NO
FIND
R-COMMITTEE
FOUND
?
YES
"RC" TO
R-FLAGG
QB
NO
FIND
GROUP-COMMITTEE
FOUND
?
YES
"GC" TO
R-FLAGG
QB
NO
FIND
GROUP-REC
FOUND
?
YES
"GP" TO
R-FLAGG
NO
SPACES
TO R-FLAGG
ARN-EXIT
QB

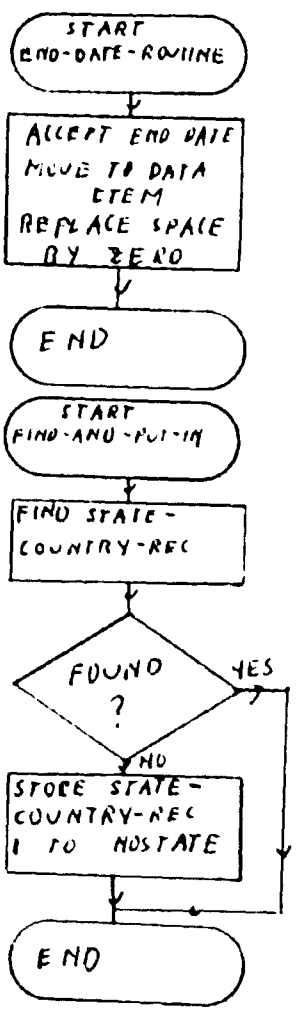

START ADDITION-OUTPUT-REPORT
MOVE NAME
WRITE REPORT
MOVE ADDRES
WRITE REPORT
END
START CHANGE-G-COMM-RESPONS
FIND-G-COMM-NAME
UNTIL CORRECT NAME FOUND OR NO MORE TO CHECK
RECORD FOUND
YES
MODIFY NAME-G-COMM-REC
NO
DISPLAY MESSAGE
END
START CHANGE-GROUP-RESPONS
FIND-GROUP-NAME
UNTIL CORRECT NAME FOUND OR NO MORE TO CHECK
RECORD FOUND ?
YES
MODIFY NAME-GROUP-REC
NO
DISPLAY MESSAGE
END
START CHANGE-OUTPUT-REPORT
MOVE NAME
WRITE REPORT
END

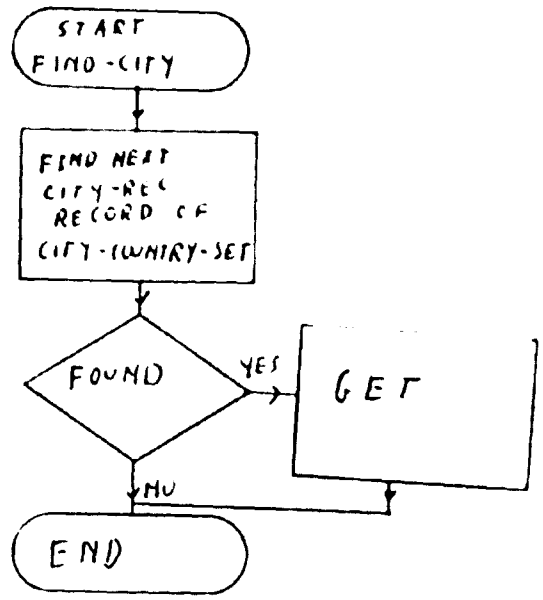

START CHANGE-R-COMM-RESPONS
UNTIL CORRECT NAME FOUND OR NO MORE TO CHECK
FIND-R-COMM-NAME
RECORD FOUND ?
YES
MODIFY NAME-R-COMM-REC
NO
DISPLAY MESSAGE
END
START CHANGE-ROSTER-RESPONS
UNTIL CORRECT NAME FOUND OR NO MORE TO CHECK
FIND-ROSTER-NAME
RECORD FOUND ?
YES
MODIFY NAME-ROSTER-REC
NO
DISPLAY MESSAGE
END
START CONNECT-ROSTER
TERMINAL MESSAGE
R
START BEGIN-ROS
ACCEPT NUMBER
SPACES INPUT ?
YES
CR-EXIT
NO
NUM-RECEIVE THRU NR-EXIT
BAD INPUT ?
YES
APPROPRIATE MESSAGE
NO
FIND ROSTER-REC
THIRD CHANCE ?
YES
C
NO
R
FOUND ?
YES
"RO" TO R-FLAG
NO
FIND R-COMMITTEE
RA
RB

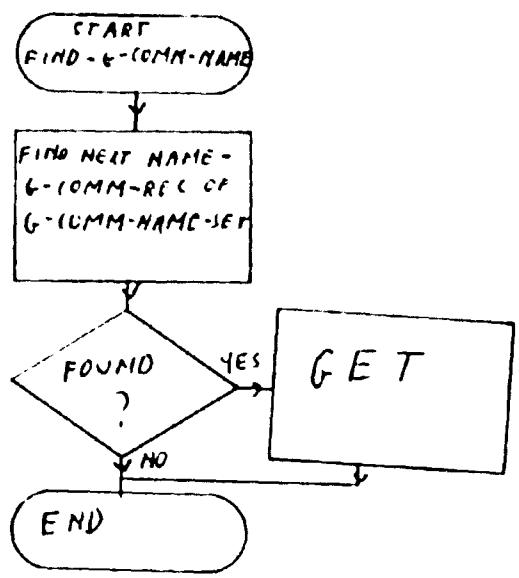

RA
FOUND ?
YES
"RC" TO R-FLAG
RB
NO
FIND GROUP-COMMITTEE
FOUND ?
YES
"GC" TO R-FLAG
RB
NO
FIND GROUP-REC
FOUND ?
YES
"GP" TO R-FLAG
RB
NO
APPROPRIATE MESSAGE
THIRD CHANCE ?
YES
CLOSE FILES
CLOSE DATABASE AREAS
STOP RUN
NO
R
RB
ACCEPT RESPONSIBILITY
R-FLAG = "RC" ?
YES
STORE NAME-R-COMM-REC
R
NO
R-FLAG = "GC" ?
YES
STORE NAME-G-COMM-REC
R
NO
STORE NAME-GROUP-REC
R

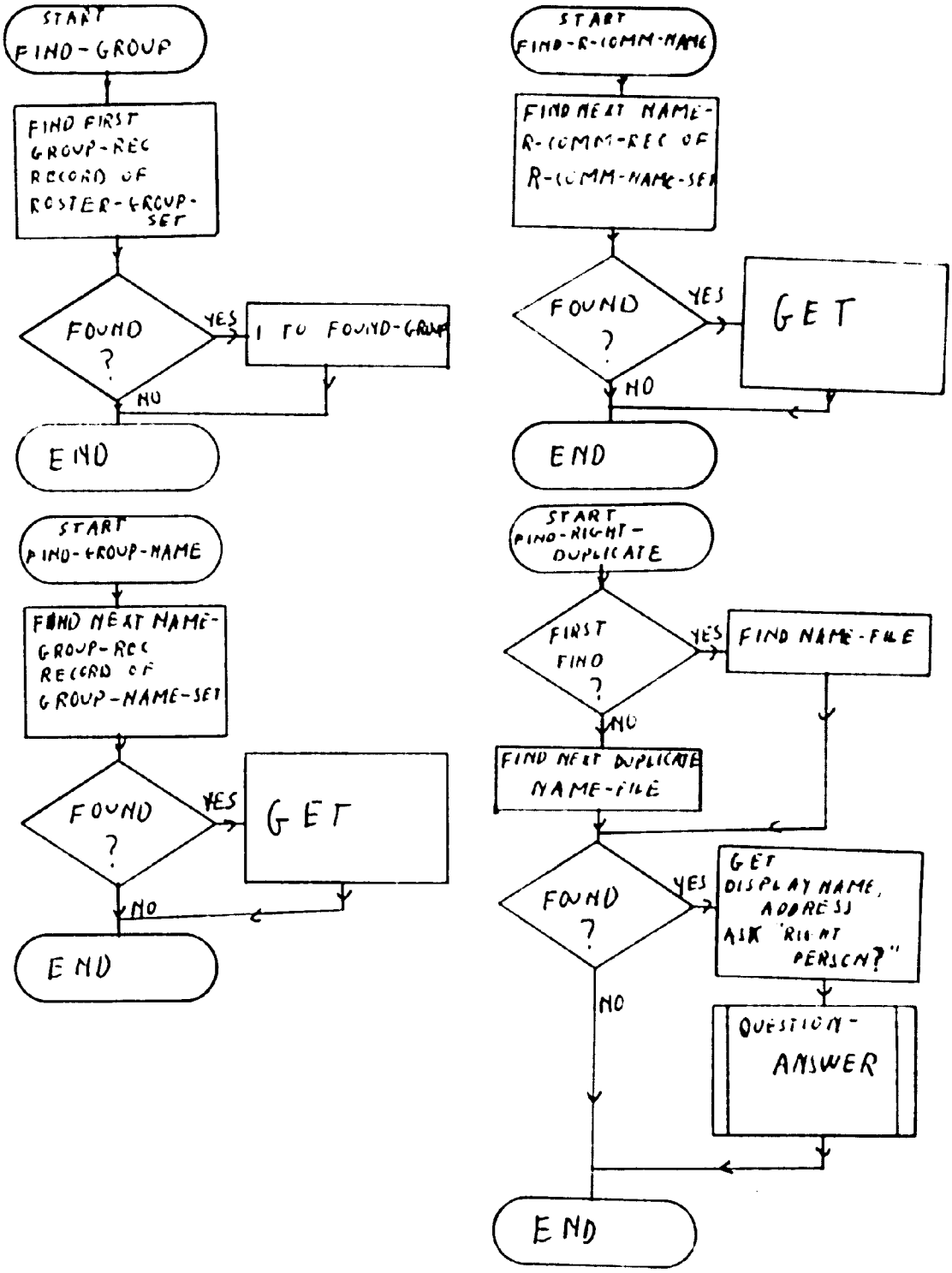

START DEL-G-COMM-NAME
FIND-G-COMM-NAME
UNTIL NAME FOUND OR NO MORE TO CHECK
FOUND ?
YES
DELETE NAME-G-COMM-REC
NO
TERMINAL MESSAGE
END
START DEL-R-COMM-NAME
FIND-R-COMM-NAME
UNTIL NAME FOUND OR NO MORE TO CHECK
FOUND ?
YES
DELETE NAME-R-COMM-REC
NO
TERMINAL MESSAGE
END
START DEL-GROUP-NAME
FIND-GROUP-NAME
UNTIL NAME FOUND OR NO MORE TO CHECK
FOUND ?
YES
DELETE NAME-GROUP-REC
NO
TERMINAL MESSAGE
END
START DEL-ROS-NAME
FIND-ROSTER-NAME
UNTIL NAME FOUND OR NO MORE TO CHECK
RECORD FOUND ?
YES
DELETE NAME-ROSTER-REC
NO
TERMINAL MESSAGE
END

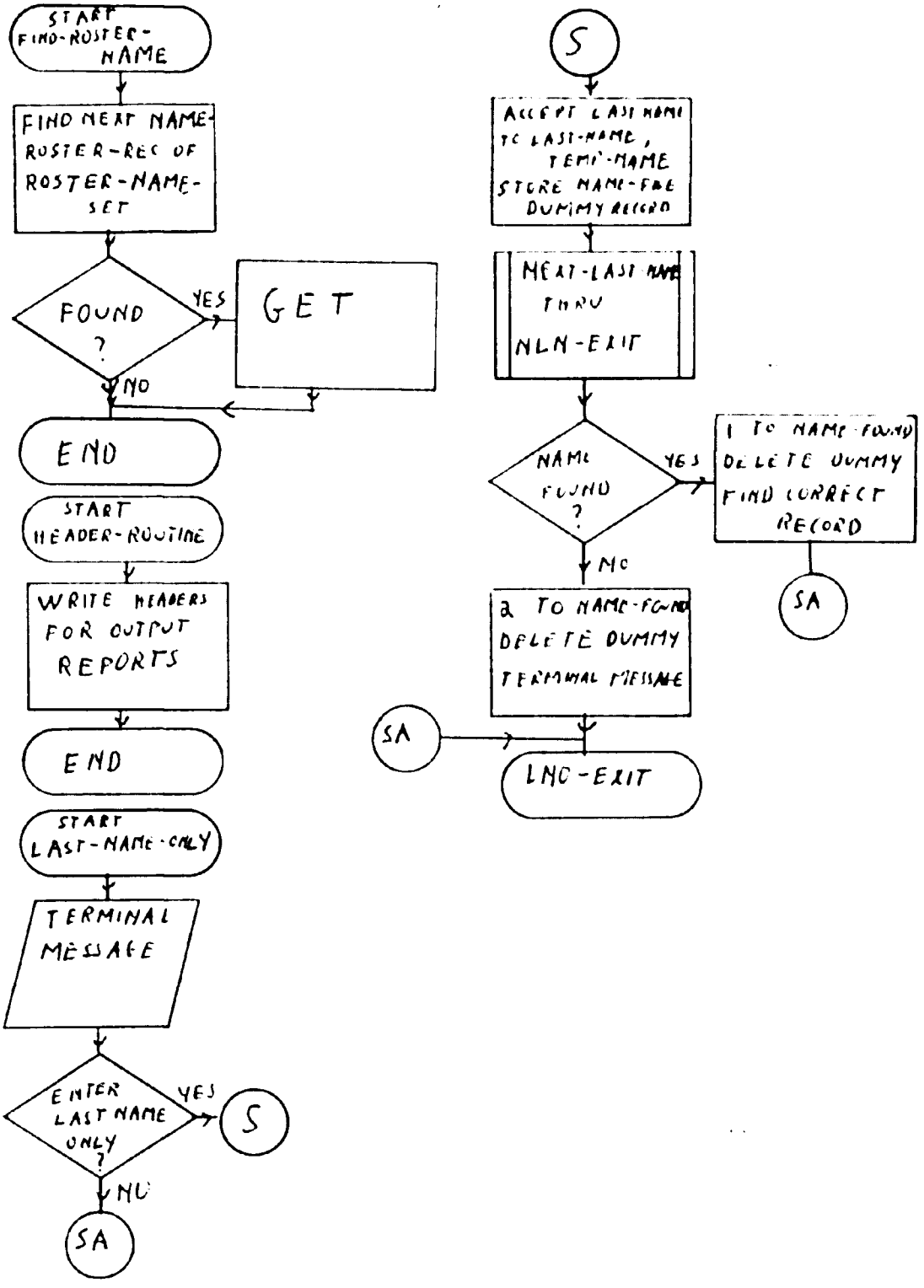

START
DELETION-REPORT-
ROUTINE
PERSON
OR
ORGANIZATION
YES
WHAT-IT-IS TO
PRINT-LINE
WRITE REPORT
NO
NAME TO
PRINT-LINE
WRITE REPORT
END

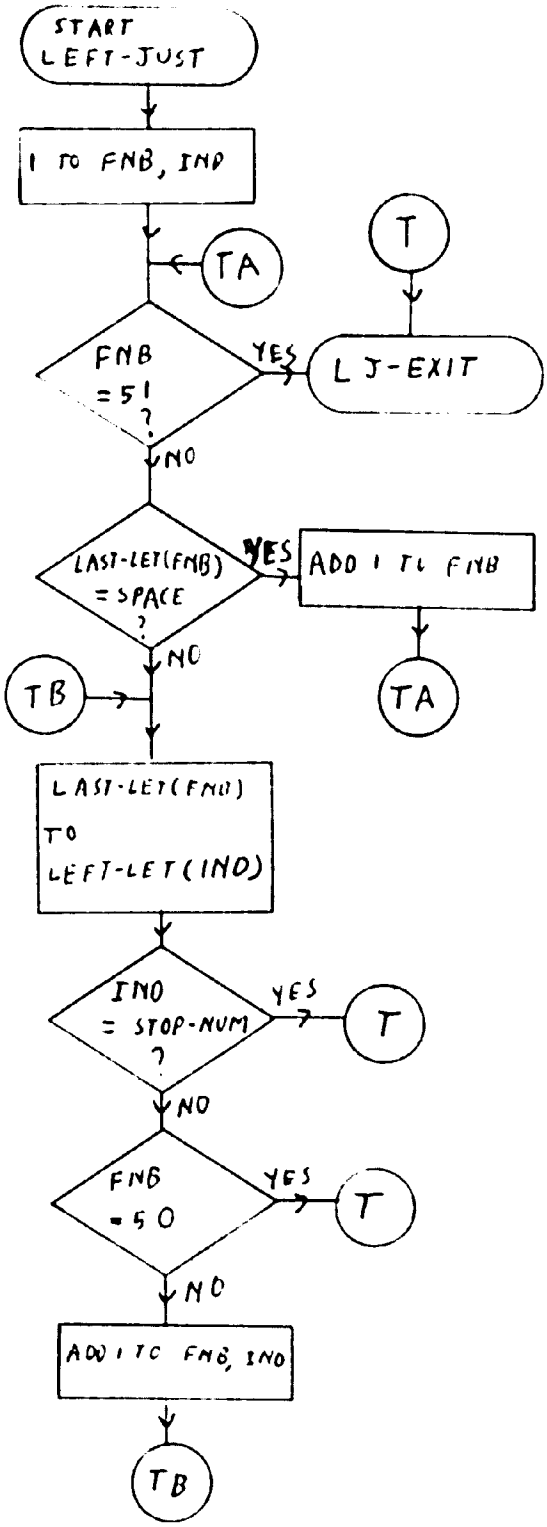

START
END-DATE-ROUTINE
ACCEPT END DATE
MOVE TO DATA
ITEM
REPLACE SPACE
BY ZERO
END
START
FIND-AND-PUT-IN
FIND STATE-
COUNTRY-REC
FOUND
?
YES
NO
STORE STATE-
COUNTRY-REC
1 TO NOSTATE
END

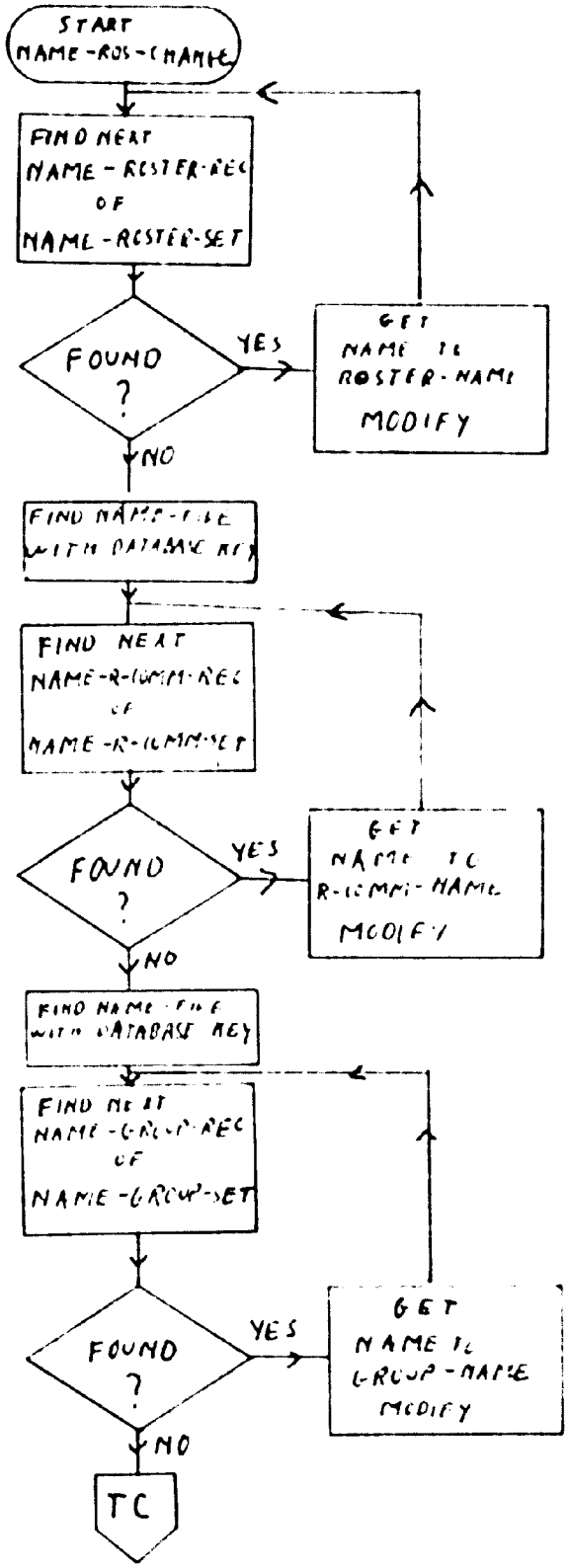

START
FIND-CITY
FIND NEXT
CITY-REC
RECORD OF
CITY-COUNTRY-SET
FOUND
YES
GET
NO
END

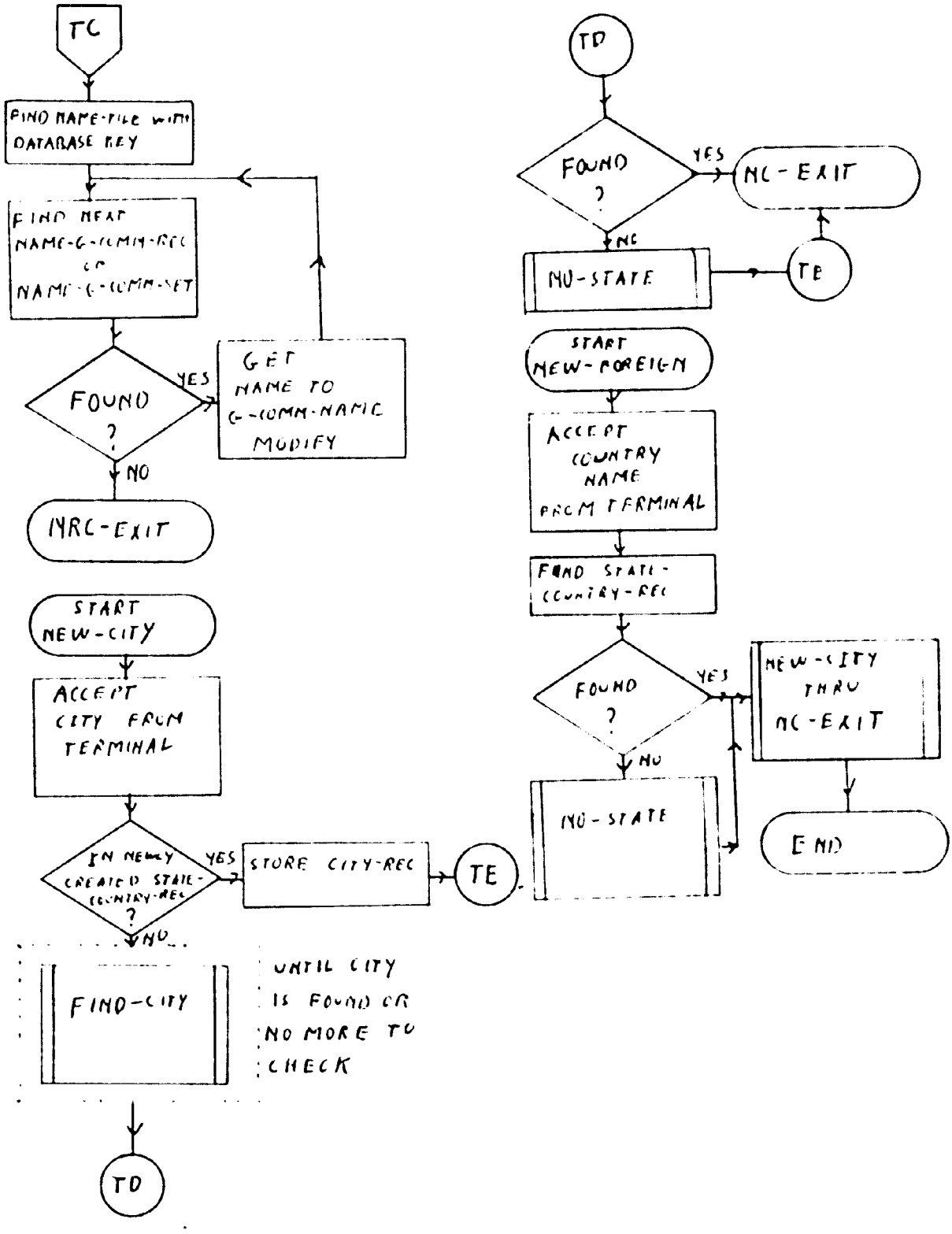

START
FIND-G-COMM-NAME
FIND NEXT NAME-
G-COMM-REC OF
G-COMM-NAME-SET
FOUND
?
YES
GET
NO
END

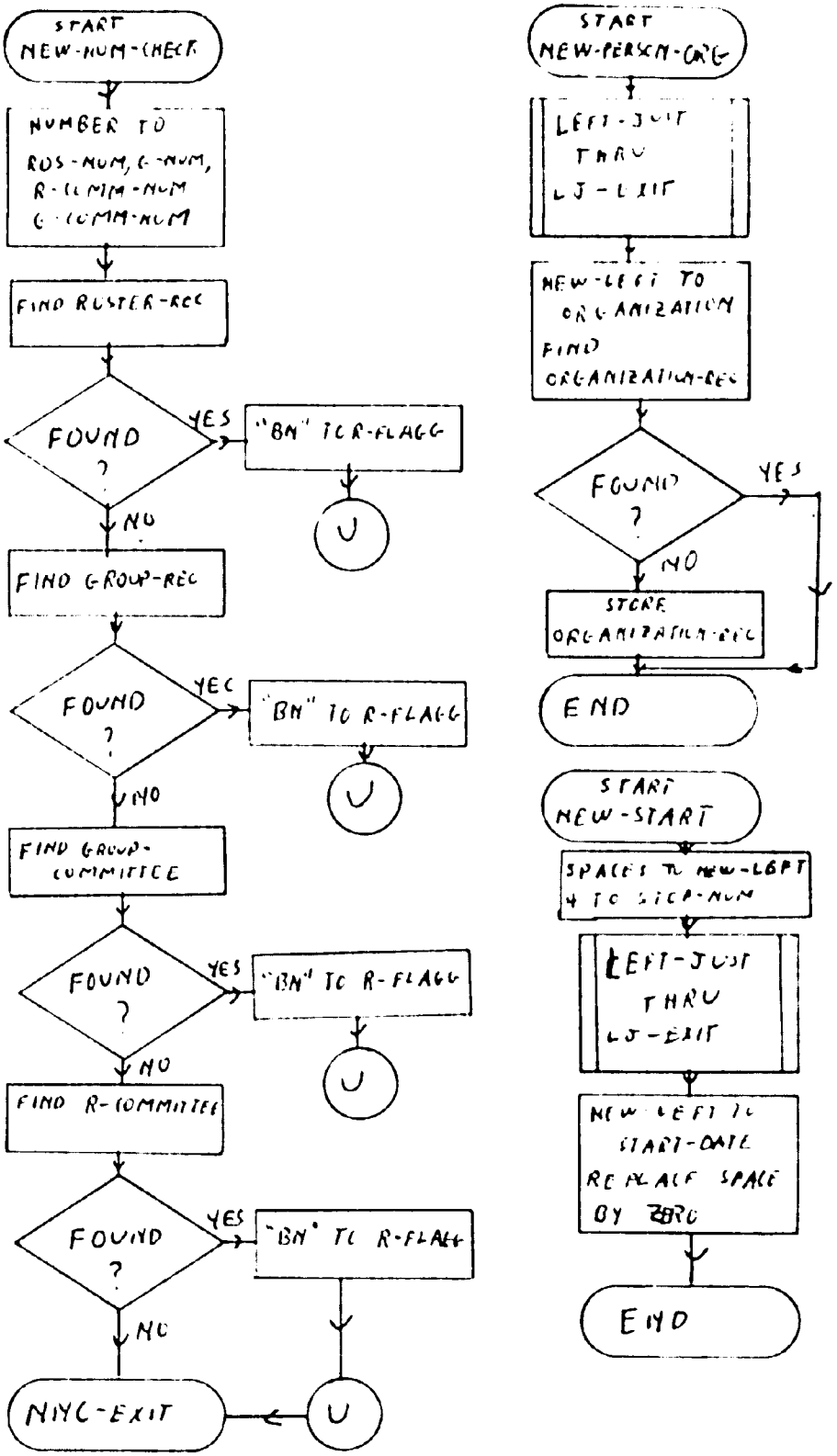

START
FIND-GROUP
FIND FIRST GROUP-REC RECORD OF ROSTER-GROUP-SET
FOUND ?
YES
1 TO FOUND-GROUP
NO
END
START
FIND-GROUP-NAME
FIND NEXT NAME-GROUP-REC RECORD OF GROUP-NAME-SET
FOUND ?
YES
GET
NO
END
START
FIND-R-COMM-NAME
FIND NEXT NAME-R-COMM-REC OF R-COMM-NAME-SET
FOUND ?
YES
GET
NO
END
START
FIND-RIGHT-DUPLICATE
FIRST FIND ?
YES
FIND NAME-FILE
NO
FIND NEXT DUPLICATE NAME-FILE
FOUND ?
YES
GET DISPLAY NAME, ADDRESS ASK "RIGHT PERSON?"
NO
QUESTION-ANSWER
END

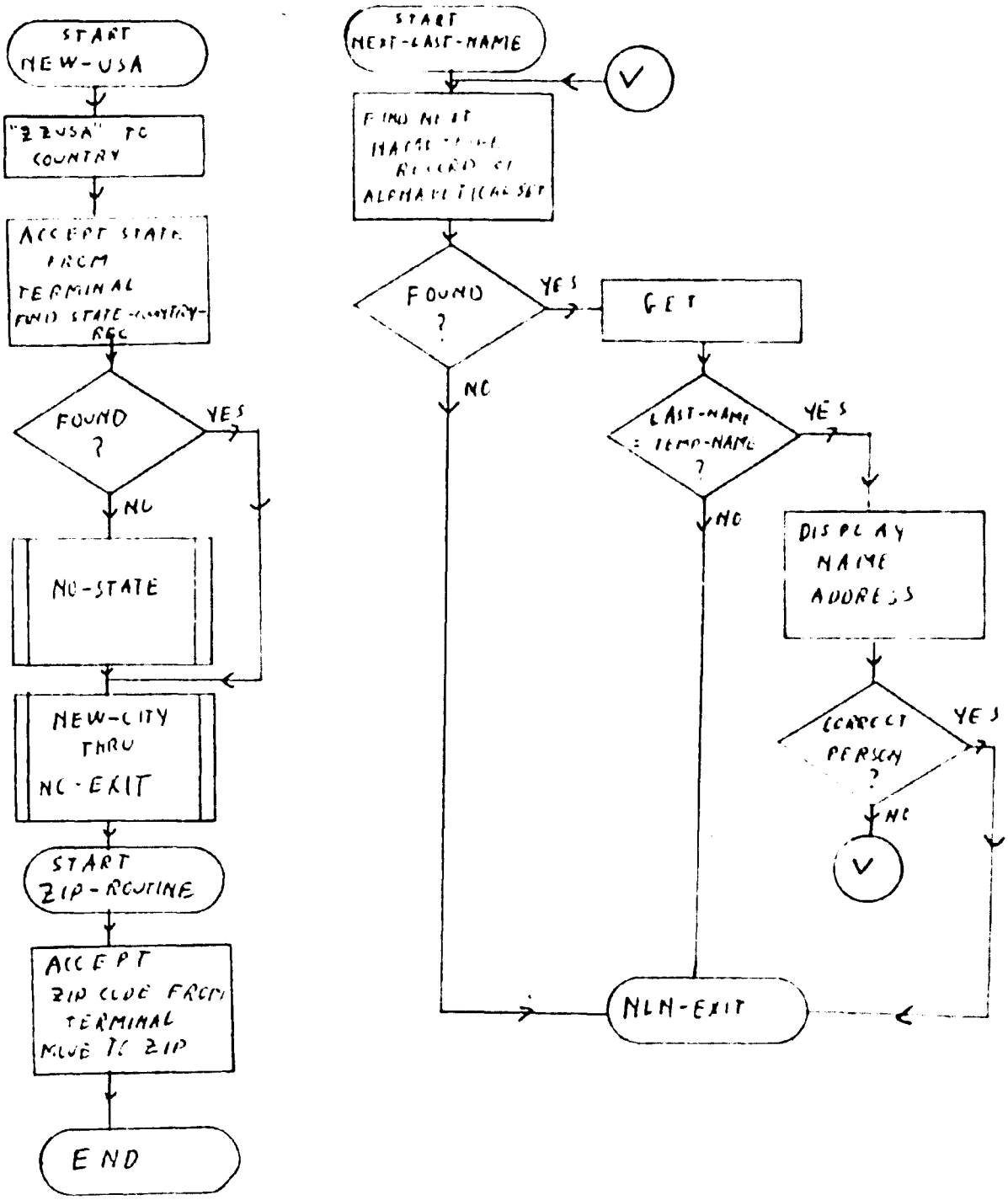

START FIND-ROSTER-NAME
FIND NEXT NAME-ROSTER-REC OF ROSTER-NAME-SET
FOUND ?
YES
GET
NO
END
START HEADER-ROUTINE
WRITE HEADERS FOR OUTPUT REPORTS
END
START LAST-NAME-ONLY
TERMINAL MESSAGE
ENTER LAST NAME ONLY ?
YES
S
NO
SA
S
ACCEPT LAST NAME TO LAST-NAME, TEMP-NAME STORE NAME-FOR DUMMY RECORD
NEXT-LAST NAME THRU NLN-EXIT
NAME FOUND ?
YES
1 TO NAME-FOUND DELETE DUMMY FIND CORRECT RECORD
SA
NO
2 TO NAME-FOUND DELETE DUMMY TERMINAL MESSAGE
SA
LNO-EXIT

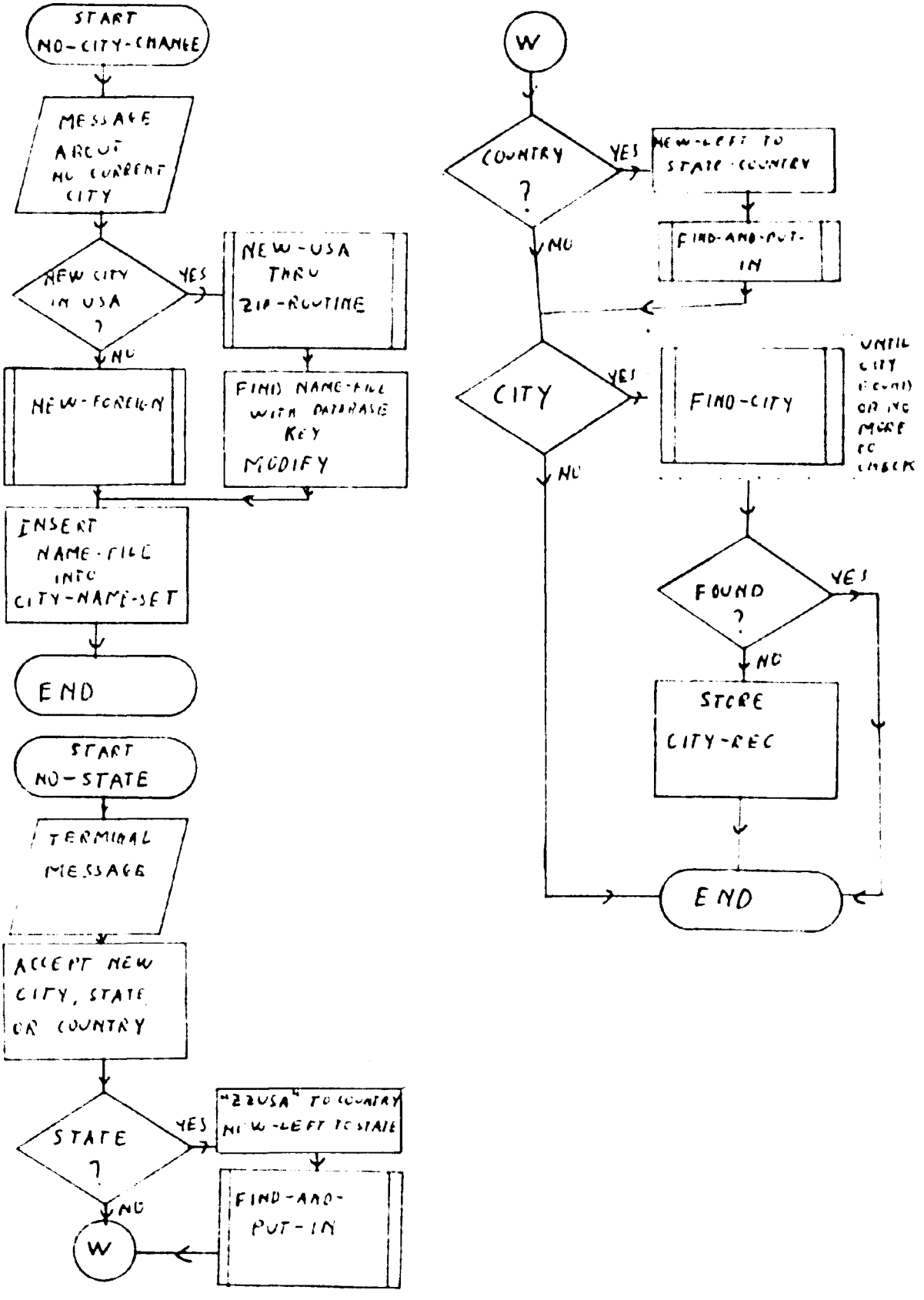

START
LEFT-JUST
1 TO FNB, IND
TA
T
FNB
=51
?
YES
LJ-EXIT
NO
LAST-LET(FNB)
= SPACE
?
YES
ADD 1 TO FNB
NO
TA
TB
LAST-LET(FNB)
TO
LEFT-LET(IND)
IND
= STOP-NUM
?
YES
T
NO
FNB
=50
YES
T
NO
ADD 1 TO FNB, IND
TB

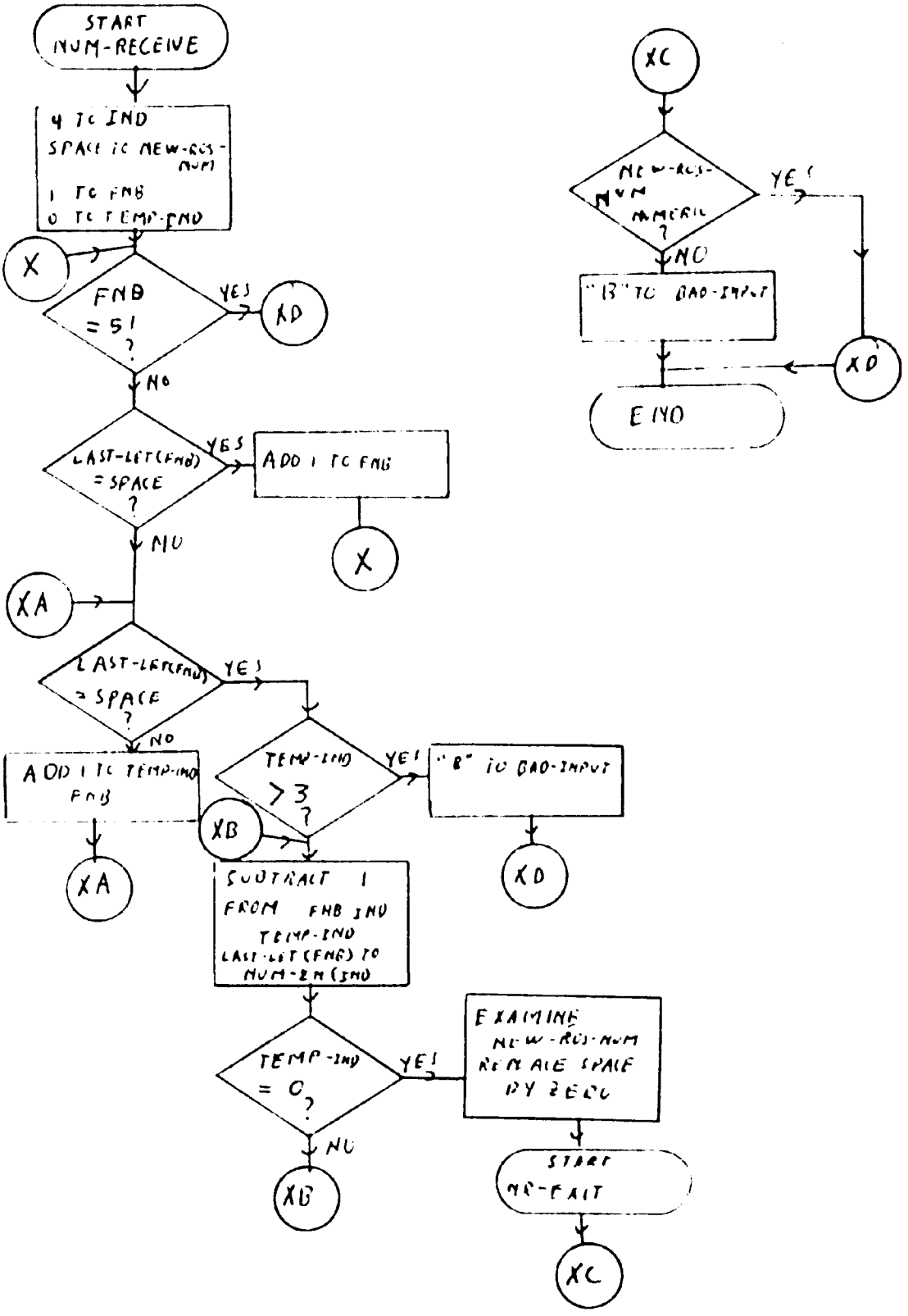

START
NAME-ROS-CHANGE
FIND NEXT
NAME-ROSTER-REC
OF
NAME-ROSTER-SET
FOUND
?
YES
GET
NAME TO
ROSTER-NAME
MODIFY
NO
FIND NAME-REC
WITH DATABASE KEY
FIND NEXT
NAME-R-COMM-REC
OF
NAME-R-COMM-SET
FOUND
?
YES
GET
NAME TO
R-COMM-NAME
MODIFY
NO
FIND NAME-REC
WITH DATABASE KEY
FIND NEXT
NAME-GROUP-REC
OF
NAME-GROUP-SET
FOUND
?
YES
GET
NAME TO
GROUP-NAME
MODIFY
NO
TC

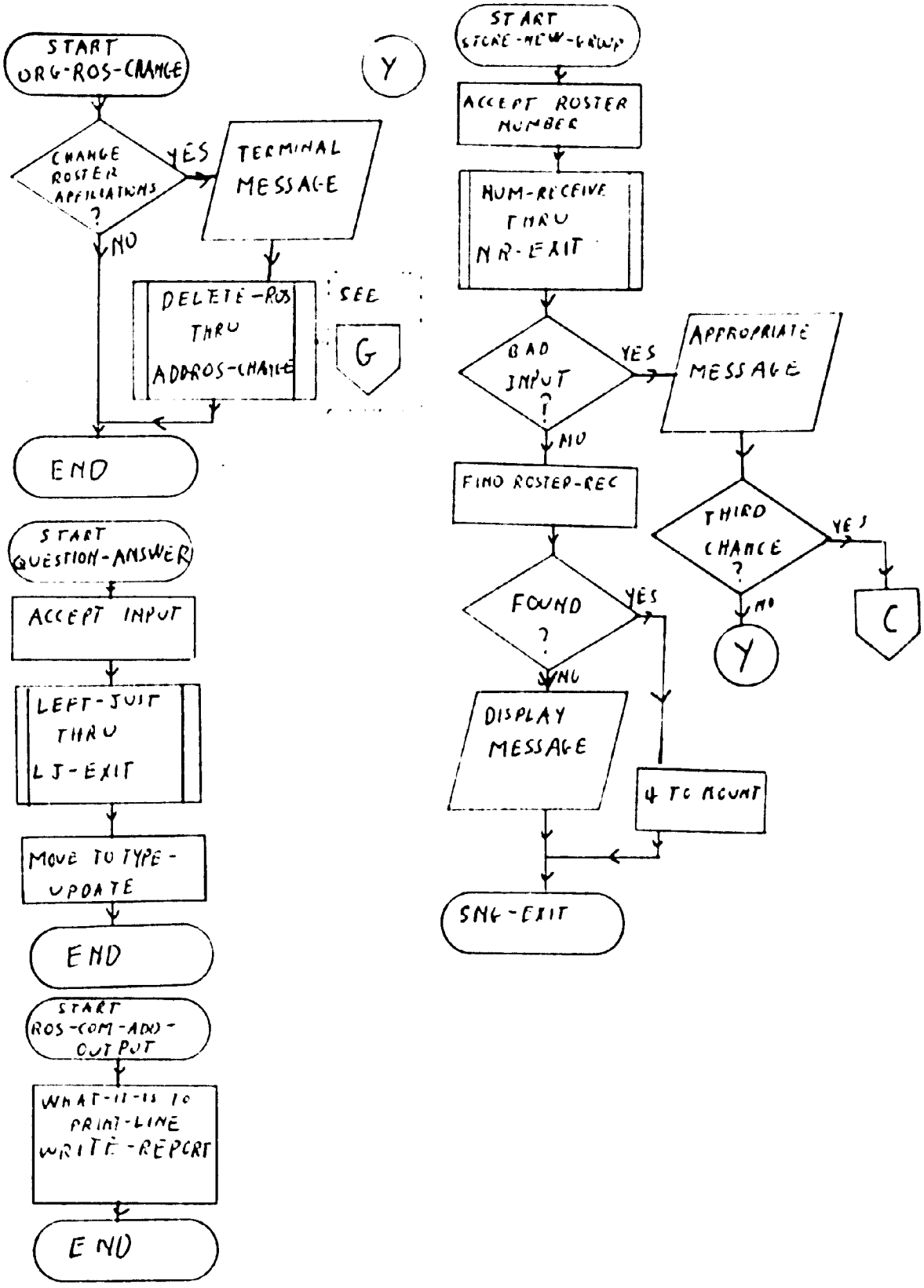

TC
FIND NAME-FILE WITH DATABASE KEY
FIND NEXT NAME-G-COMM-REC OR NAME-G-COMM-SET
FOUND ?
YES
GET NAME TO G-COMM-NAME MODIFY
NO
NRC-EXIT
START NEW-CITY
ACCEPT CITY FROM TERMINAL
IN NEWLY CREATED STATE-COUNTRY-REC ?
YES
STORE CITY-REC
TE
NO
FIND-CITY
UNTIL CITY IS FOUND OR NO MORE TO CHECK
TD
TD
FOUND ?
YES
NC-EXIT
NO
NO-STATE
TE
START NEW-FOREIGN
ACCEPT COUNTRY NAME FROM TERMINAL
FIND STATE-COUNTRY-REC
FOUND ?
YES
NEW-CITY THRU NC-EXIT
NO
NO-STATE
END


START
NEW-NUM-CHECK
NUMBER TO
ROS-NUM, C-NUM,
R-COMM-NUM
C-COMM-NUM
FIND ROSTER-REC
FOUND ?
YES
"BN" TO R-FLAGG
NO
FIND GROUP-REC
FOUND ?
YES
"BN" TO R-FLAGG
NO
FIND GROUP-COMMITTEE
FOUND ?
YES
"BN" TO R-FLAGG
NO
FIND R-COMMITTEE
FOUND ?
YES
"BN" TO R-FLAGG
NO
NNC-EXIT
START
NEW-PERSON-ORG
LEFT-JUST
THRU
LJ-EXIT
NEW-LEFT TO
ORGANIZATION
FIND
ORGANIZATION-REC
FOUND ?
YES
NO
STORE
ORGANIZATION-REC
END
START
NEW-START
SPACES TO NEW-LEFT
4 TO SICA-NUM
LEFT-JUST
THRU
LJ-EXIT
NEW-LEFT TO
START-DATE
REPLACE SPACE
BY ZERO
END


START
NEW-USA
"2"USA" TO
COUNTRY
ACCEPT STATE
FROM
TERMINAL
FIND STATE-COUNTRY-
REC
FOUND
?
YES
NO
NO-STATE
NEW-CITY
THRU
NO-EXIT
START
ZIP-ROUTINE
ACCEPT
ZIP CODE FROM
TERMINAL
MOVE TO ZIP
END
START
NEXT-LAST-NAME
FIND NEXT
RECORD OF
ALPHABETICAL SET
FOUND
?
YES
NO
GET
LAST-NAME
= TEMP-NAME
?
YES
NO
DISPLAY
NAME
ADDRESS
CORRECT
PERSON
?
YES
NO
NLN-EXIT


START NO-CITY-CHANGE
MESSAGE ABOUT NO CURRENT CITY
NEW CITY IN USA ?
YES
NO
NEW-USA THRU ZIP-ROUTINE
NEW-FOREIGN
FIND NAME-FILE WITH DATABASE KEY MODIFY
INSERT NAME-FILE INTO CITY-NAME-SET
END
START NO-STATE
TERMINAL MESSAGE
ACCEPT NEW CITY, STATE OR COUNTRY
STATE ?
YES
"22USA" TO COUNTRY NEW-LEFT TO STATE
FIND-AND-PUT-IN
NO
W
W
COUNTRY ?
YES
NEW-LEFT TO STATE-COUNTRY
FIND-AND-PUT-IN
NO
CITY ?
YES
FIND-CITY
UNTIL CITY FOUND OR NO MORE TO CHECK
NO
FOUND ?
YES
NO
STORE CITY-REC
END


START NUM-RECEIVE
4 TO IND
SPACE TO NEW-ROS-NUM1
1 TO FNB
0 TO TEMP-IND
X
FNB = 5 ?
YES
XD
NO
LAST-LET(FNB) = SPACE ?
YES
ADD 1 TO FNB
X
NO
XA
LAST-LET(FNB) = SPACE ?
YES
NO
ADD 1 TO TEMP-IND FNB
XA
TEMP-IND > 3 ?
YES
"8" TO BAD-INPUT
XD
XB
SUBTRACT 1 FROM FNB IND TEMP-IND
LAST-LET(FNB) TO NUM-EM(IND)
TEMP-IND = 0 ?
YES
EXAMINE NEW-ROS-NUM REPLACE SPACE BY ZERO
START NS-EXIT
XC
NO
XB
XC
NEW-ROS-NUM NUMERIC ?
YES
NO
"13" TO BAD-INPUT
XD
END


START ORG-ROS-CHANGE
CHANGE ROSTER APPLICATIONS ?
YES
TERMINAL MESSAGE
NO
DELETE-ROS THRU ADDROS-CHANGE
SEE
G
Y
END
START QUESTION-ANSWER
ACCEPT INPUT
LEFT-JUST THRU LJ-EXIT
MOVE TO TYPE-UPDATE
END
START ROS-COM-AND-OUTPUT
WHAT-IT-IS TO PRINT-LINE WRITE-REPORT
END
START STORE-NEW-GROUP
ACCEPT ROSTER NUMBER
NUM-RECEIVE THRU NR-EXIT
BAD INPUT ?
YES
APPROPRIATE MESSAGE
NO
FIND ROSTER-REC
THIRD CHANCE ?
YES
NO
C
Y
FOUND ?
YES
NO
DISPLAY MESSAGE
4 TO ACCUM
SNG-EXIT

APPENDIX E

DOCUMENTATION AND FLOWCHART FOR INQUIRY PROGRAM

DOCUMENTATION FOR THE INQUIRY PROGRAM

The inquiry program (READDB.CBL) is a COBOL program designed to allow a user to interrogate the data base concerning the data listed for an individual member. The interrogation is to be done interactively. The program accesses the data base in retrieval mode, hence can only read what is in the data base, and cannot affect the contents of the data base.

The program starts by opening the areas of the data base in retrieval mode, and then displays a message describing what the program does, and how to answer questions.

The next paragraph, called REQUESTS, actually starts the questions asked of the user. The routine QUESTION-ANSWER THRU QA-EXIT (line 10700) is used to move the answers to questions to the data item TYPE-UPDATE, which is then tested for the actual input. The first question asked is whether the record to be checked is for a person member or for an organizational member. If the input is anything other than P or O the program ceases execution. If P is input (or anything starting with P), PERSON-MEM THRU PM-EXIT (line 12100) is performed to find the record of the correct member. If O

is input (or anything starting with O), then ORGAN-MEM THRU OM-EXIT (line 17100) is performed to find the correct record. If the member's record cannot be found, a message to that effect is displayed, and the program branches back to REQUESTS.

If the record is found, the program asks if the user wishes to see the associated geographical data. If the answer is yes, the routine GEO-DISPLAY THRU GO-EXIT (line 26000) is performed. The next question is whether the user wishes to see the associated organization. If so, the routine ORG-DISPLAY (line 10610) is performed.

The user is then asked if the phone number, the start date, and the end date are to be shown. Finally the user is asked if a list of committees is to be displayed. If the user wants the list, ROSTER-LIST THRU RL-EXIT (line 22000) is performed. Then the program branches back to REQUESTS.

ALPHABETICAL LIST OF ROUTINES

GEO-DISPLAY THRU GO-EXIT (line 26000):

This routine first tries to find the city of the member. If the attempt is unsuccessful, the routine displays a message to that effect, and ends. If the city's record is found, the routine finds the country,

and checks to see if the country is the United States. If it is, the country, state, and city are displayed. If the country is not the United States, the country and city are displayed.

LEFT-JUST THRU LJ-EXIT (line 20700):

This routine is used to left justify input names. It looks for the first non-blank character, and starting with that one, moves the input, character by character, to the data item NEW-LEFT.

ORG-DISPLAY (line 10610):

This routine tries to find the member's organization. If the attempt is unsuccessful, a message to that effect is displayed. If the attempt is successful, the organization's name is displayed.

ORGAN-MEM THRU OM-EXIT (line 17100):

This routine asks for the name of the organizational member whose record is to be interrogated. LEFT-JUST THRU LJ-EXIT (line 20700) is used to left justify the name. The name is moved to NAIM, and the routine then tries to find the associated record. If the record cannot be found, the routine ends. If the record is found, the name and address are displayed, and the user is asked to verify that the correct record has been found. If it has. the routine ends.

If the user indicates that the correct record has not been found, the routine looks for a record of a member with the same name. If there is no such record, a message to that effect is displayed, and the routine ends. Otherwise, the above process is repeated until either the user indicates the correct record has been found, or there are no more records to check.

PERSON-MEM THRU PM-EXIT (line 12100):

This routine acts just like ORGAN-MEM THRU OM-EXIT. The only difference is that the member's name is input in three parts, first the last name, then the first name, and then the middle name.

QUESTION-ANSWER THRU QA-EXIT (line 10700):

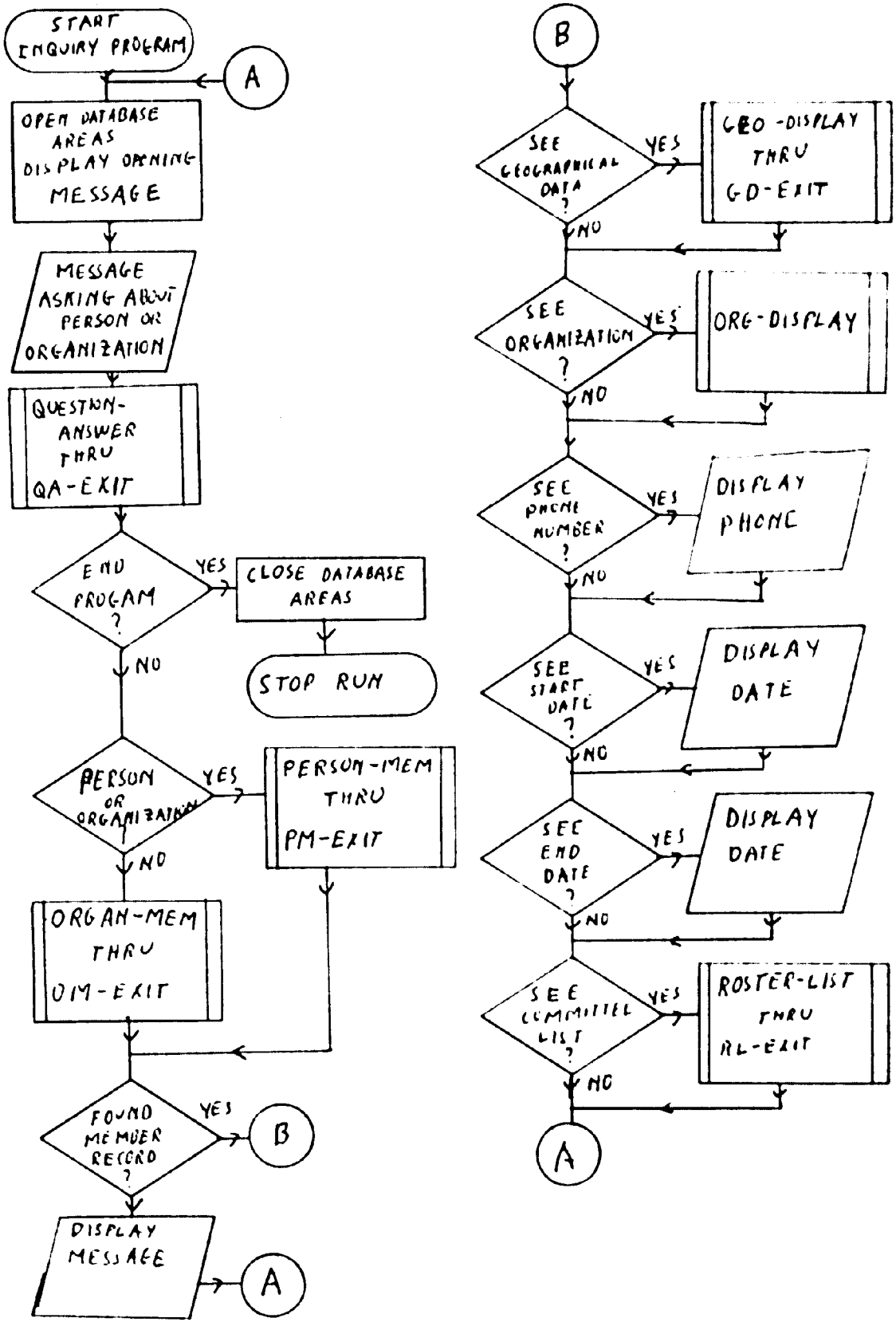
This routine is used to move answers to questions posed by the machine to the data item TYPE-UPDATE. The routine looks for the first non-blank input character, and if it finds one, moves the character to TYPE-UPDATE.

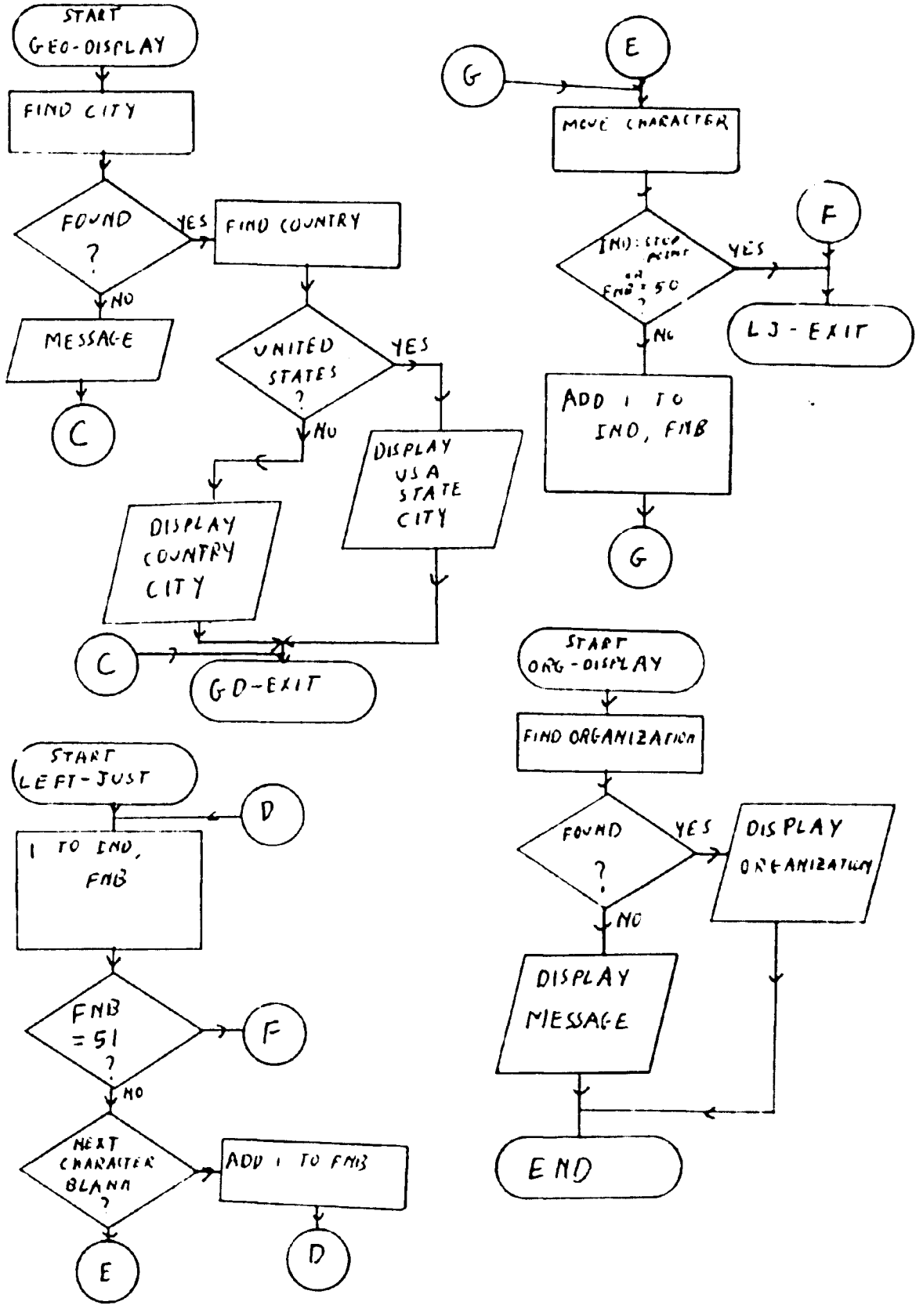
ROSTER-LIST THRU RL-EXIT (line 22000):

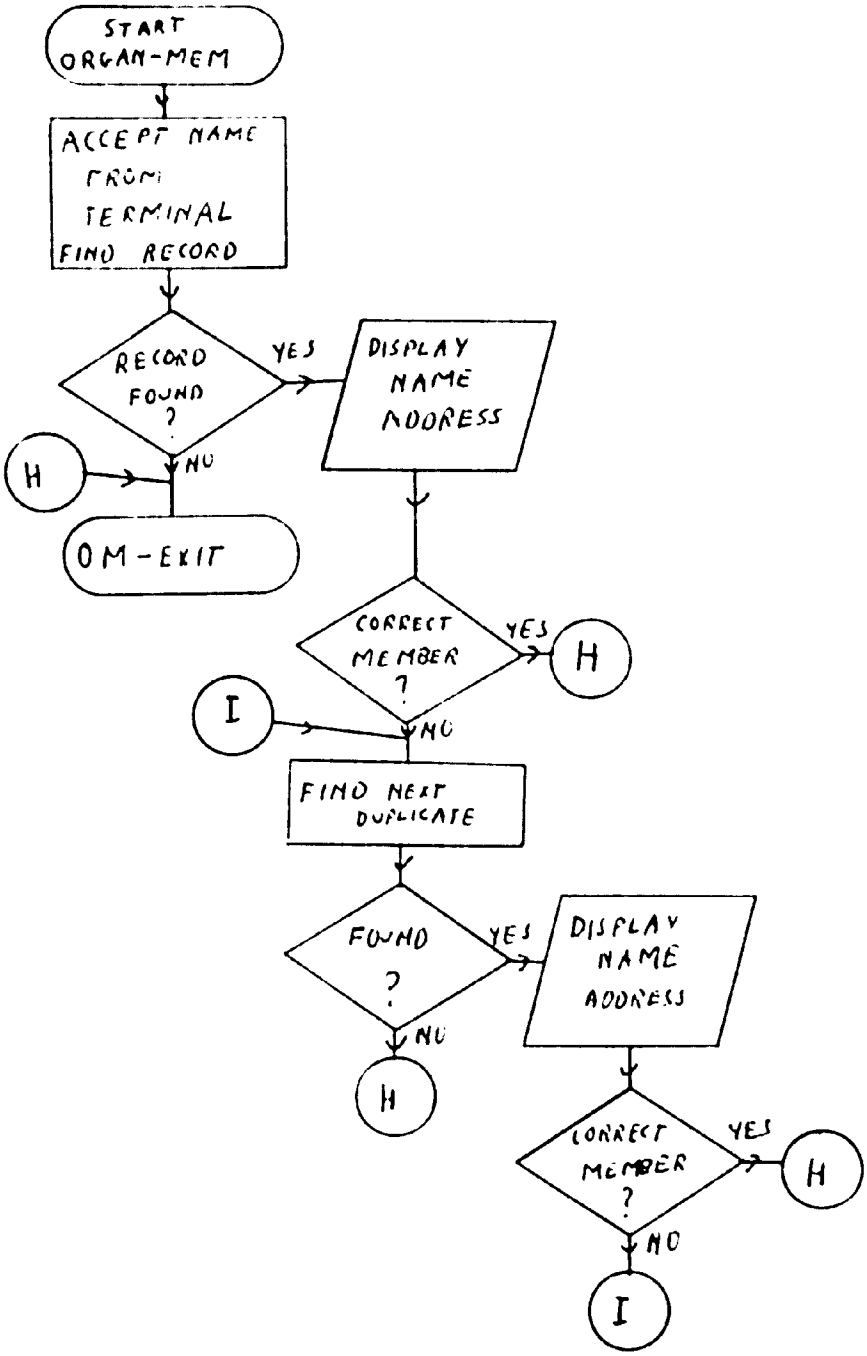
This routine is used to display the description of any roster, group, or committee that the member is on, along with the member's responsibility on that roster, group, or committee.

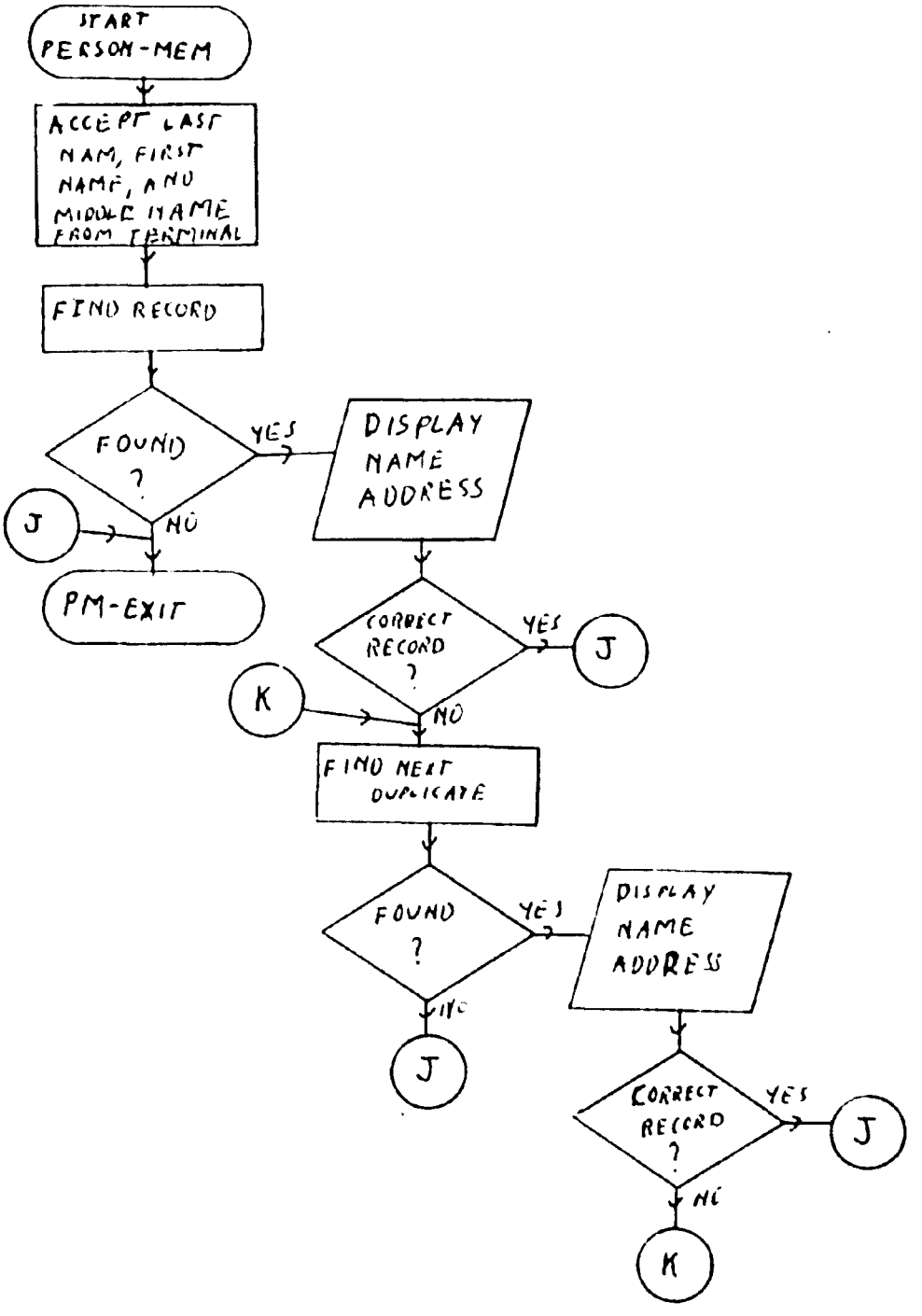
The routine starts by looking for the member's name connected to a roster. If it finds such, it

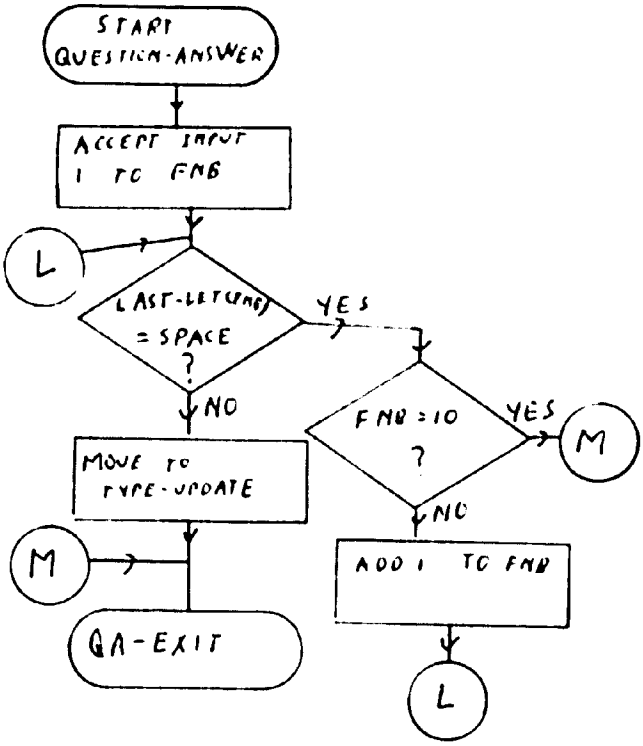
accesses the roster record, displays the description of the roster and the member's responsibility, and looks for the next roster connection. The above process repeats until the routine finds all rosters that the member is connected with. Then it does the same for roster committees, groups, and group committees in that order.

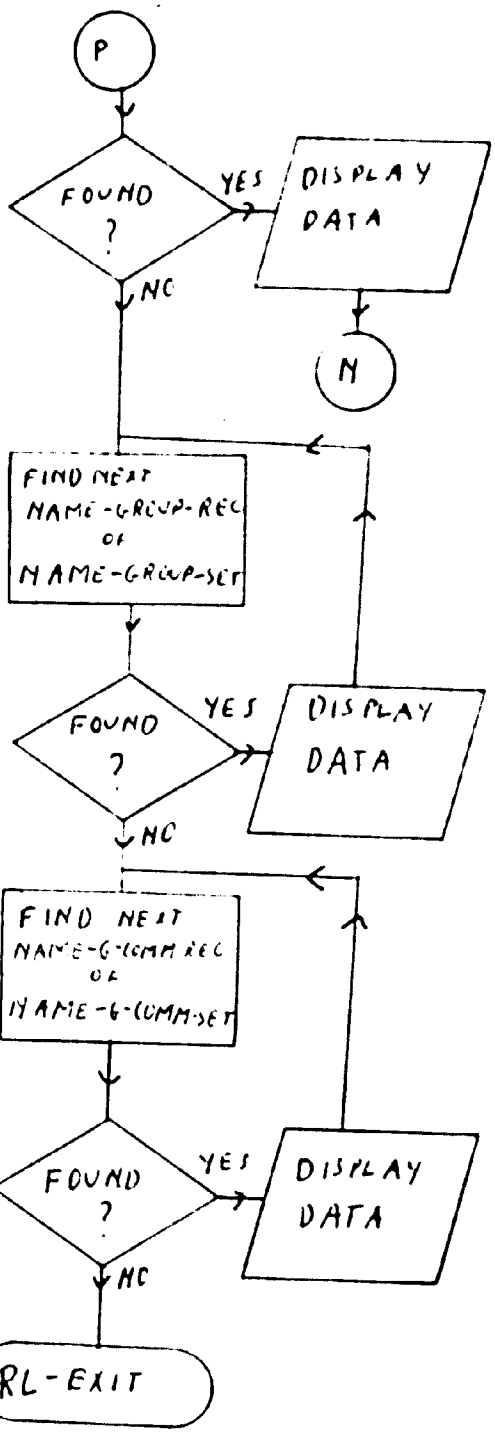

START ENQUIRY PROGRAM
A
OPEN DATABASE AREAS DISPLAY OPENING MESSAGE
MESSAGE ASKING ABOUT PERSON OR ORGANIZATION
QUESTION-ANSWER THRU QA-EXIT
END PROGAM ?
YES
CLOSE DATABASE AREAS
STOP RUN
NO
PERSON OR ORGANIZATION
YES
PERSON-MEM THRU PM-EXIT
NO
ORGAN-MEM THRU OM-EXIT
FOUND MEMBER RECORD ?
YES
B
DISPLAY MESSAGE
A
B
SEE GEOGRAPHICAL DATA ?
YES
GEO-DISPLAY THRU GD-EXIT
NO
SEE ORGANIZATION ?
YES
ORG-DISPLAY
NO
SEE PHONE NUMBER ?
YES
DISPLAY PHONE
NO
SEE START DATE ?
YES
DISPLAY DATE
NO
SEE END DATE ?
YES
DISPLAY DATE
NO
SEE COMMITTEE LIST ?
YES
ROSTER-LIST THRU RL-EXIT
NO
A

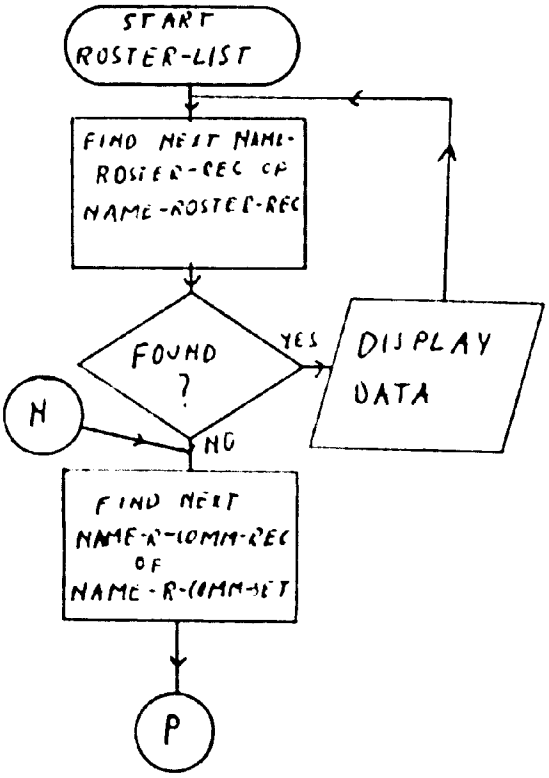

START
GEO-DISPLAY
FIND CITY
FOUND ?
YES
NO
FIND COUNTRY
MESSAGE
C
UNITED STATES ?
YES
NO
DISPLAY USA STATE CITY
DISPLAY COUNTRY CITY
C
GD-EXIT
G
E
MOVE CHARACTER
IND: STOP POINT OR FNB = 50 ?
YES
NO
F
LJ-EXIT
ADD 1 TO IND, FNB
G
START
ORG-DISPLAY
FIND ORGANIZATION
FOUND ?
YES
NO
DISPLAY ORGANIZATION
DISPLAY MESSAGE
END
START
LEFT-JUST
D
1 TO END, FNB
FNB = 51 ?
F
NO
NEXT CHARACTER BLANK ?
ADD 1 TO FNB
E
D


START
ORGAN-MEM
ACCEPT NAME
FROM
TERMINAL
FIND RECORD
RECORD
FOUND
?
YES
NO
DISPLAY
NAME
ADDRESS
H
OM-EXIT
CORRECT
MEMBER
?
YES
H
I
NO
FIND NEXT
DUPLICATE
FOUND
?
YES
NO
DISPLAY
NAME
ADDRESS
H
CORRECT
MEMBER
?
YES
H
NO
I


START
PERSON-MEM
ACCEPT LAST NAM, FIRST NAME, AND MIDDLE NAME FROM TERMINAL
FIND RECORD
FOUND ?
YES
NO
DISPLAY NAME ADDRESS
J
PM-EXIT
CORRECT RECORD ?
YES
NO
J
K
FIND NEXT DUPLICATE
FOUND ?
YES
NO
DISPLAY NAME ADDRESS
J
CORRECT RECORD ?
YES
NO
J
K


START QUESTION-ANSWER
ACCEPT INPUT 1 TO FNB
L
LAST-LETTER = SPACE ?
YES
NO
FNB = 10 ?
YES
M
NO
ADD 1 TO FNB
L
MOVE TO TYPE-UPDATE
M
QA-EXIT


P
FOUND ?
YES
DISPLAY DATA
N
NO
FIND NEXT NAME-GROUP-REC OF NAME-GROUP-SET
FOUND ?
YES
DISPLAY DATA
NO
FIND NEXT NAME-G-COMM REC OF NAME-G-COMM-SET
FOUND ?
YES
DISPLAY DATA
NO
RL-EXIT


START ROSTER-LIST
FIND NEXT NAME-ROSTER-REC OF NAME-ROSTER-REC
FOUND ?
YES
DISPLAY DATA
N
NO
FIND NEXT NAME-R-COMM-REC OF NAME-R-COMM-SET
P

APPENDIX F

DOCUMENTATION AND FLOWCHART FOR INTERFACE PROGRAM

DOCUMENTATION FOR THE INTERFACE PROGRAM

The interface program (TEMPFI.CBL) is a COBOL program designed to give selected lists culled from the data base. The lists are by rosters, committees, and/or groups. The program can also supply an alphabetical list of the entire membership, or give a listing of all rosters, broken up into groups and committees when relevant, together with members. The program accesses the data base in retrieval mode only. The desired list is on a disc file called ROSLIS.DAT. The program is interactive only to the extent of accepting which rosters, committees, and/or groups should be on the list from the user.

The program starts by opening the output file and the data base areas. It then displays a message explaining the user options, and performs ACCEPT-ROS-NUM THRU ARN-EXIT (line 34700) to receive the user input.

The program then checks to see which option the user has selected. If ABC was entered ALPHABETICAL-LIST THRU ALPH-EXIT (line 14200) is performed to give an alphabetical list of the membership with associated data. If ALL was entered, the program finds the first roster record in the appropriate data base area (this is

done to provide a focal point for finding the next roster). It then performs ROSTER-MOVE THRU ALL-EXIT (see ALL-ROSTERS) to process that roster, and then performs ALL-ROSTERS THRU ALL-EXIT (line 21700) to process the remaining rosters. If ROS was entered ONLY-ROS THRU OR-EXIT (line 41800) is performed to give a list of the rosters, groups, and committees.

If neither ABC, ALL, nor ROS was entered, a selected list of rosters, groups, and/or committees was entered and stored in the table ROS-STUFF. NUM-ROS-IN contains the total number of entries made. ROS-OUT (n) contains the number of the nth entry, and FLAGG (n) contains the kind of the nth entry.

The entries are processed one at a time. When all are processed, the program ends. If an entry is a roster ROSTER-MOVE THRU ALL-EXIT (see ALL-ROSTERS) is performed. If an entry is a roster committee R-COMM-MOVE THRU INTERNAL-EXIT (see ROSTER-WITH-COMM) is performed. If an entry is a group GROUP-MOVE THRU INTERNAL-CHECK-2 (see ROSTER-WITH-GROUPS) is performed. Finally, if an entry is a group committee G-COMM-MOVE THRU INTERNAL-CHECK-1 (see ROSTER-WITH-GROUPS) is performed.

ALPHABETICAL LIST OF ROUTINES

ACCEPT-ROS-NUM THRU ARN-EXIT (line 34700):

This routine accepts input from the user, and checks its validity. It starts by asking for input. If blanks are input, the routine ends. If anything else is input, the routine performs NUM-RECEIVE THRU NR-EXIT (line 34600) to move the input to the data item NEW-ROS-NUM. If the input is ALL or ABC, the routine ends. With any other input, the routine checks the data item BAD-INPUT to see if the input is in the right form. If it is, the routine checks the rosters, roster committees, groups, and group committees, in that order, looking for the desired entity. When it is found, its number and kind are stored, and the routine asks for the next number. If at any time the user input is not good, an error message is displayed, and the user is given the chance to reenter the input. If three successive bad inputs are entered, the program will terminate.

ALL-ROSTERS THRU ALL-EXIT (line 21700):

This routine is used when the user has entered ALL. It also contains the coding used to process a roster which was entered as part of a selected list.

The routine starts by finding the next roster record. If there is no next roster record, the routine

ends. If there is another record, paragraph ROSTER-MOVE starts the actual processing of the roster.

ROSTER-MOVE saves the data base key of the roster, moves the roster data to the output line, and writes the output. Then it checks to see if the roster has groups. If the roster has groups, the routine ROSTER-WITH-GROUPS THRU RWG-EXIT (line 25300) is performed to complete processing of the roster.

If there are no groups, the routine checks to see if the roster has committees. If it does, ROSTER-WITH-COMM THRU RWC-EXIT (line 31500) is performed to complete processing of the roster.

If there are neither committees nor groups, paragraph ROSTER-ALONE is used to process the members of the roster. The next name on the roster list is found. If there is no next name, the roster record is found via its data base key (so that the next roster may be found) and processing of the roster is completed. If a name is found, its data base key is saved, its member record is found, and NAME-FILE-MOVE THRU WRITE-IT-IN (see ALPHABETICAL-LIST) is performed to process the member. Then the roster record and the list name are found with their data base keys (so that the next name on the list may be found) and the routine branches back to ROSTER-ALONE.

ALPHABETICAL-LIST THRU ALPH-EXIT (line 14200):

This routine is used when ABC is entered by the user. It also contains the coding used to process a member whose name appears on a roster, group, or committee list.

The routine starts by finding the next member record. If there are no more member records, the routine ends. If there is another one, paragraph NAME-FILE-MOVE starts the processing of the record.

NAME-FILE-MOVE saves the data base key of the member record, moves the member information to the output line, finds the member's geographic and organizational data and moves that to the output line. The paragraph WRITE-IT-IN is used to write the output line. Following that, the routine branches back to ALPHABETICAL-LIST.

NUM-RECEIVE THRU NR-EXIT (line 39600):

This routine is used to move user input data to the data item NEW-ROS-NUM. It considers only the first three input characters. It moves them character by character, starting with the right-most character, to NEW-ROS-NUM. Then it replaces blanks by zeros, and checks the input to see if it is numeric.

ONLY-ROS THRU OR-EXIT (line 41800):

This routine is used to produce a list of the

rosters, groups, and committees without members. It starts by finding the first roster in TALL-AREA, and performing ROSTER-MOVE (see ALL-ROSTERS) to write the roster's record. Then the routine branches to COMM-CHECK to see if the roster has committees. If not, the routine branches to GROUP-CHECK.

If the roster has committees, the routine successively finds their records and performs R-COMM-MOVE (see ROSTER-WITH-COMM) to write the records. When all the committees of that roster have been processed, the routine goes back to NEXT-ROS to start processing the next roster record.

GROUP-CHECK checks to see if the roster has groups. If not, the routine branches back to NEXT-ROS to start processing the next roster. If the roster has groups, GROUP-MOVE is performed to process the record of the first group. Then the routine branches to NEXT-COMM-G to process the records of the committees in the group. When all the committees in the group have been processed, the routine branches to NEXT-GP to start processing the next group of the roster, and the above process is repeated. When all the groups have been processed, the routine goes back to NEXT-ROS to begin processing of the next roster. The routine ends when all the rosters have been processed.

ROSTER-WITH-COMM THRU RWC-EXIT (line 31500):

This routine is used to process a roster which is broken up into committees. It also contains the coding used to process a roster committee entered by the user. In general, it takes a committee, writes the committee details, writes the details for each member of the committee, and goes on to the next committee.

The routine finds the next committee of the roster. If there is no next committee, the routine ends. If there is another committee, paragraph R-COMM-MOVE starts the processing.

R-COMM-MOVE saves the data base key of the committee, moves the committee details to the output line, and writes the output line. Then it finds the next name on the committee list. If there is no next name, the routine branches to INTERNAL-EXIT. If there is another name, its data base key is saved, its member record is found, and NAME-FILE-MOVE THRU WRITE-IT-IN (see ALPHABETICAL-LIST) is performed. Then the list name and the committee are found with their data base keys, and the routine branches back to get the next name.

After INTERNAL-EXIT, the roster and the committee records are found with their data base keys, and the routine branches back to ROSTER-WITH-COMM.

ROSTER-WITH-GROUPS THRU RWG-EXIT (line 25300):

This routine processes a roster which is broken up into groups. It also contains the coding to process a group and a group committee. Its general strategy is to process a group followed by all the members whose names are on the group list. Then it successively processes each committee of the group followed by the members of the committee. When a group has no more committees, the routine goes on to the next group.

The routine starts by finding the next group contained within the roster. If there is no next group, the roster record is found with its data base key, and the routine ends. If there is another group, paragraph GROUP-MOVE starts its processing.

GROUP-MOVE starts by saving the data base key of the group, moving the group details to the output line, and writing the output. Then the next name on the group list is found. If there is no next name, the routine branches to GROUP-COMMITTEES. If there is another name, its data base key is saved, its member record is found, and NAME-FILE-MOVE THRU WRITE-IT-IN (see ALPHABETICAL-LIST) is performed. Then the group and the list name are found with their data base keys, and the routine branches back for the next group name.

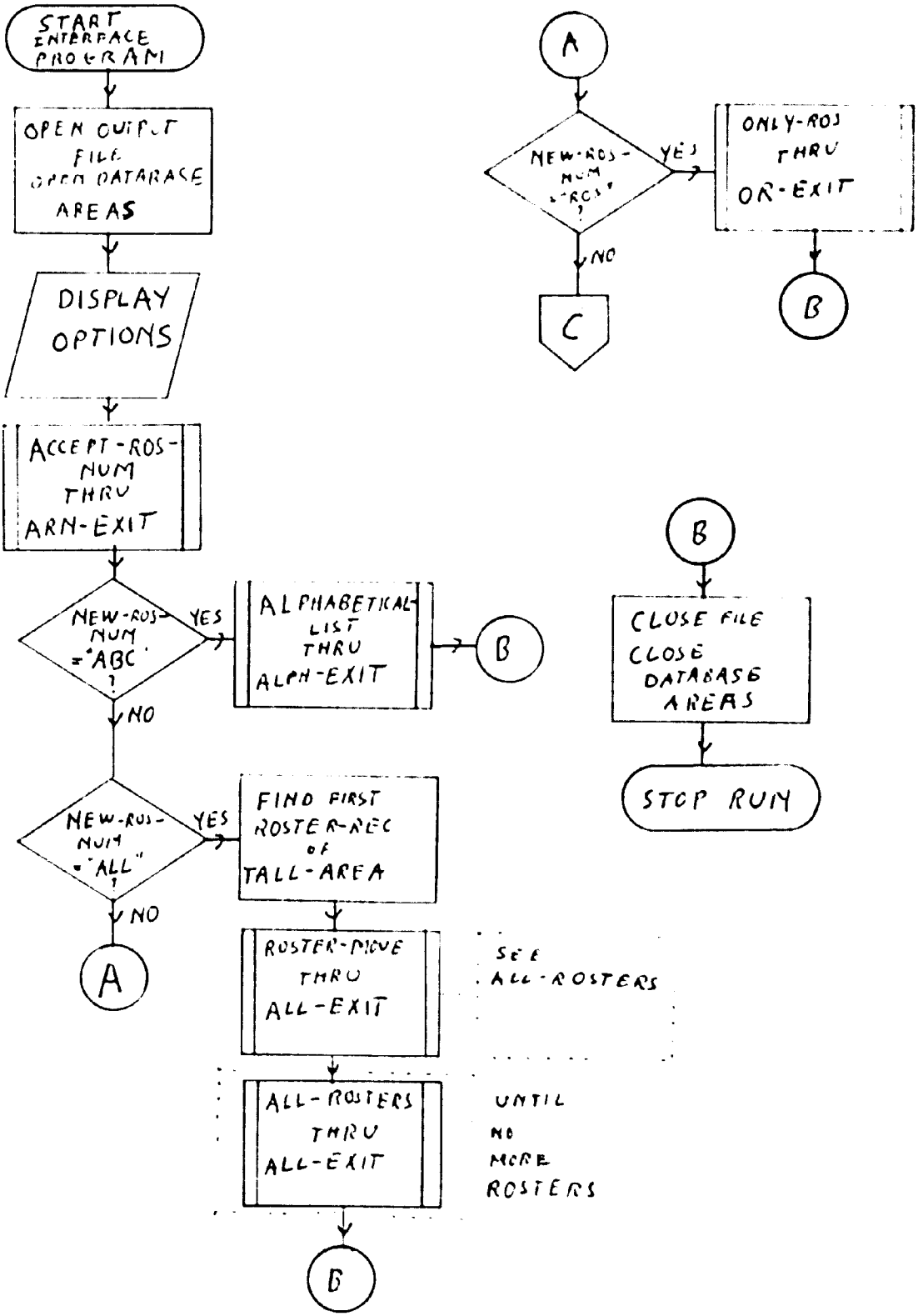
GROUP-COMMITTEES finds the next group committee

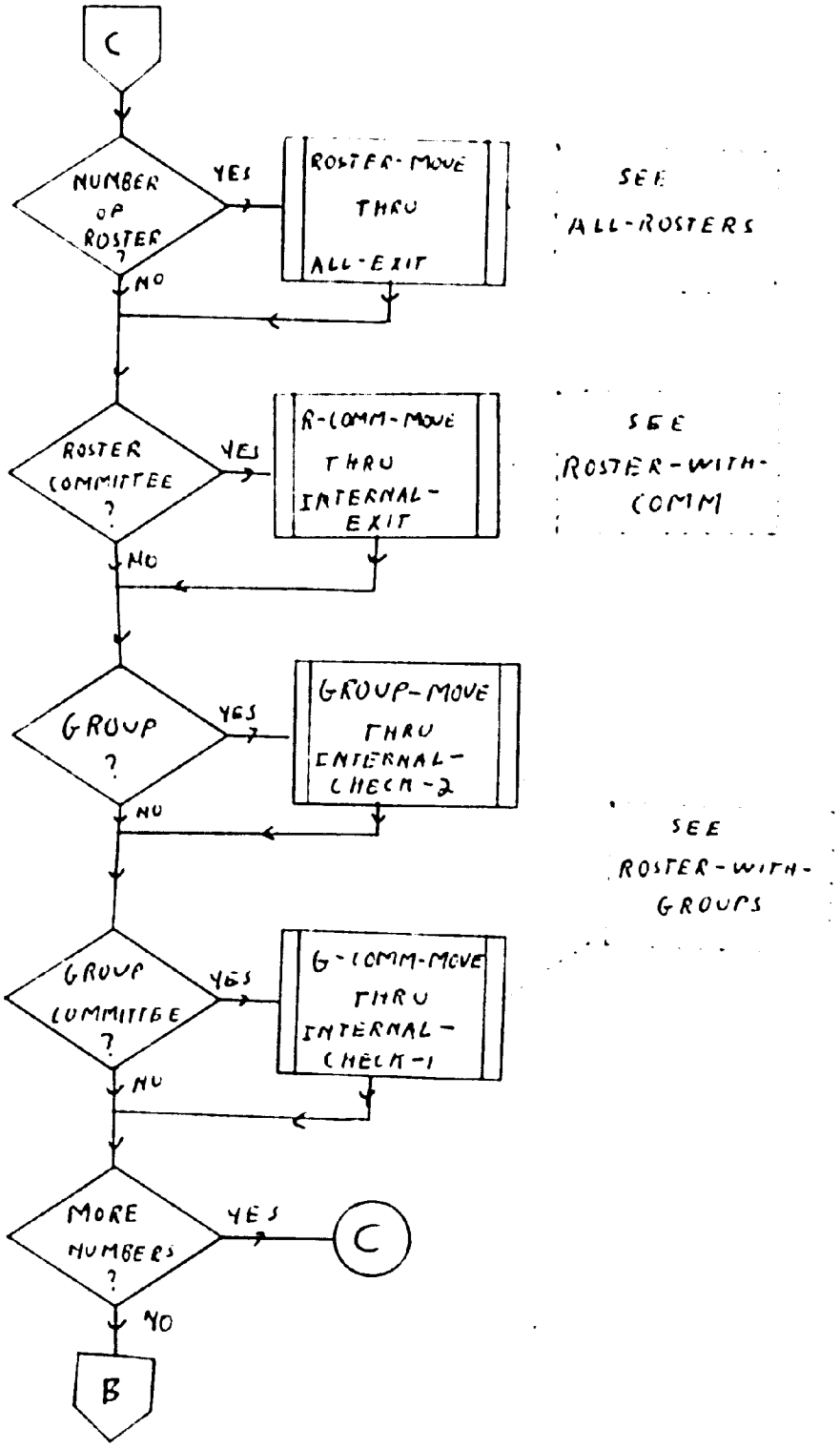
within the group. If there are no more committees, the program branches to INTERNAL-CHECK-2. If there is another committee, paragraph G-COMM-MOVE starts.

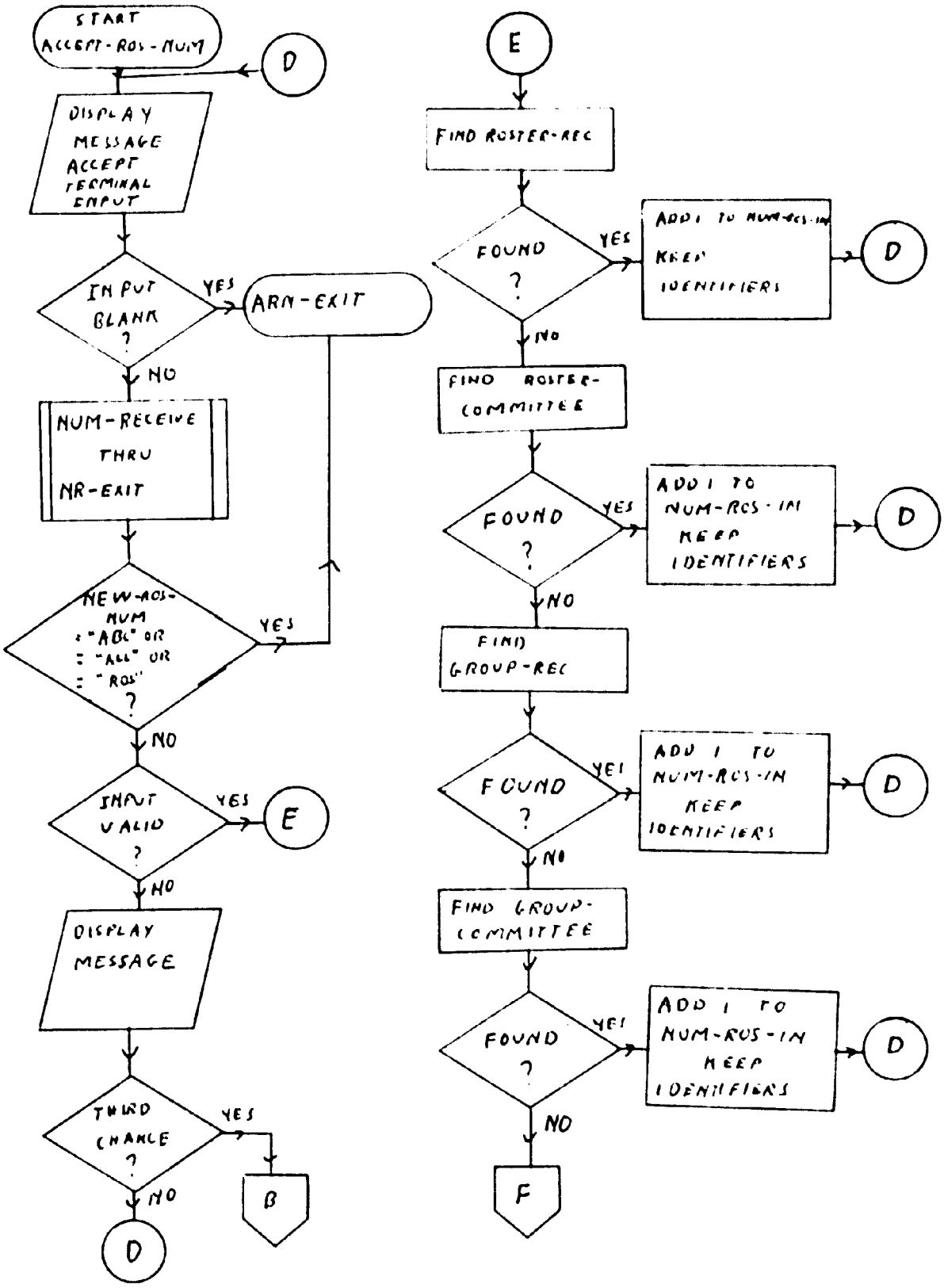
G-COMM-MOVE saves the data base key of the committee, moves the committee details to the output line, and writes the output. Then it finds the next name on the committee list. If there are no more names on the list, the program branches to INTERNAL-CHECK-1. If there is another name, its data base key is saved, its member record is found, and NAME-FILE-MOVE THRU WRITE-IT-IN (see ALPHABETICAL-LIST) is performed. Then the committee and the list name are found with their data base keys, and the routine branches back for the next name.

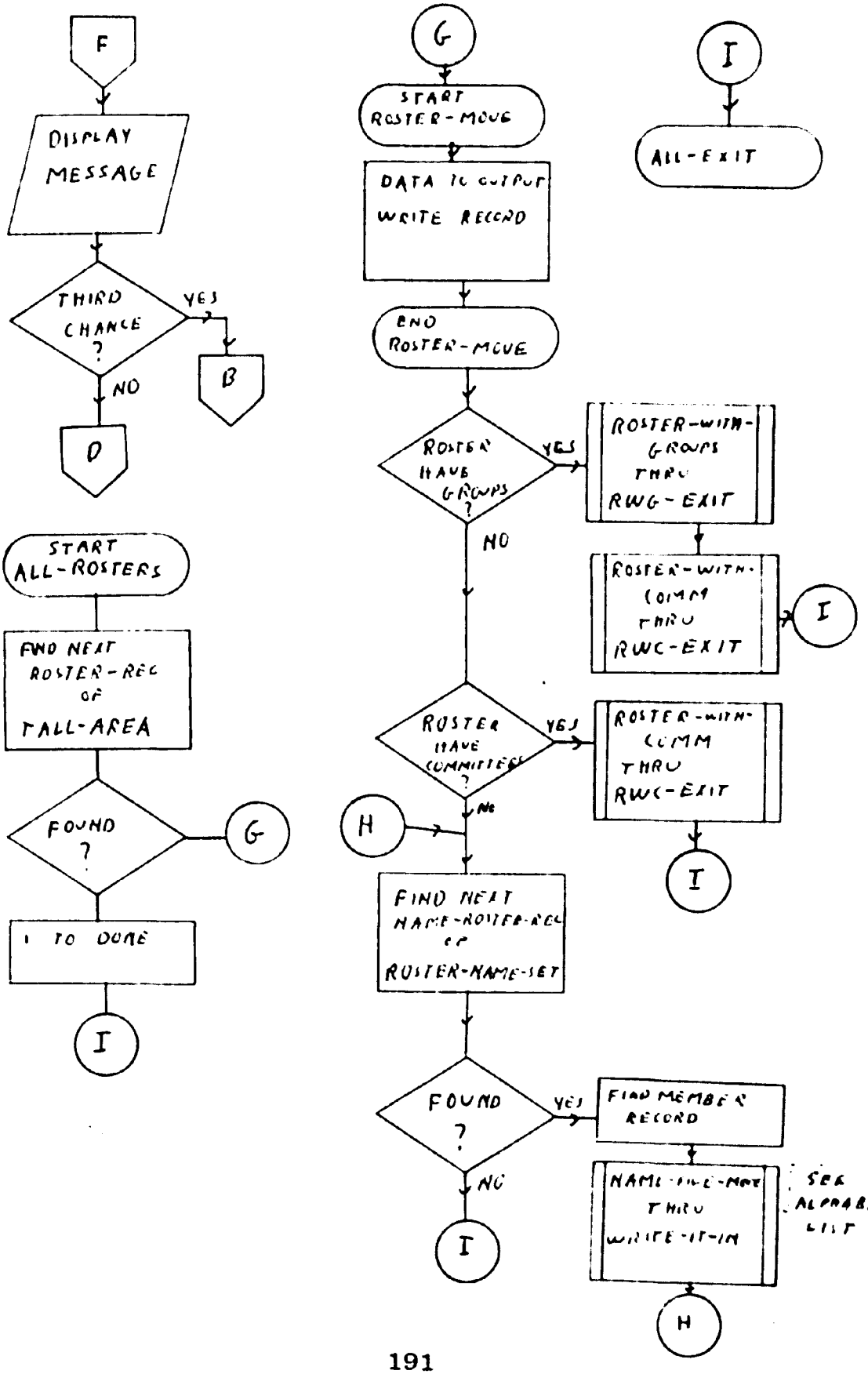
After INTERNAL-CHECK-1, the group and the committee are found with their data base keys, and processing returns to GROUP-COMMITTEES.

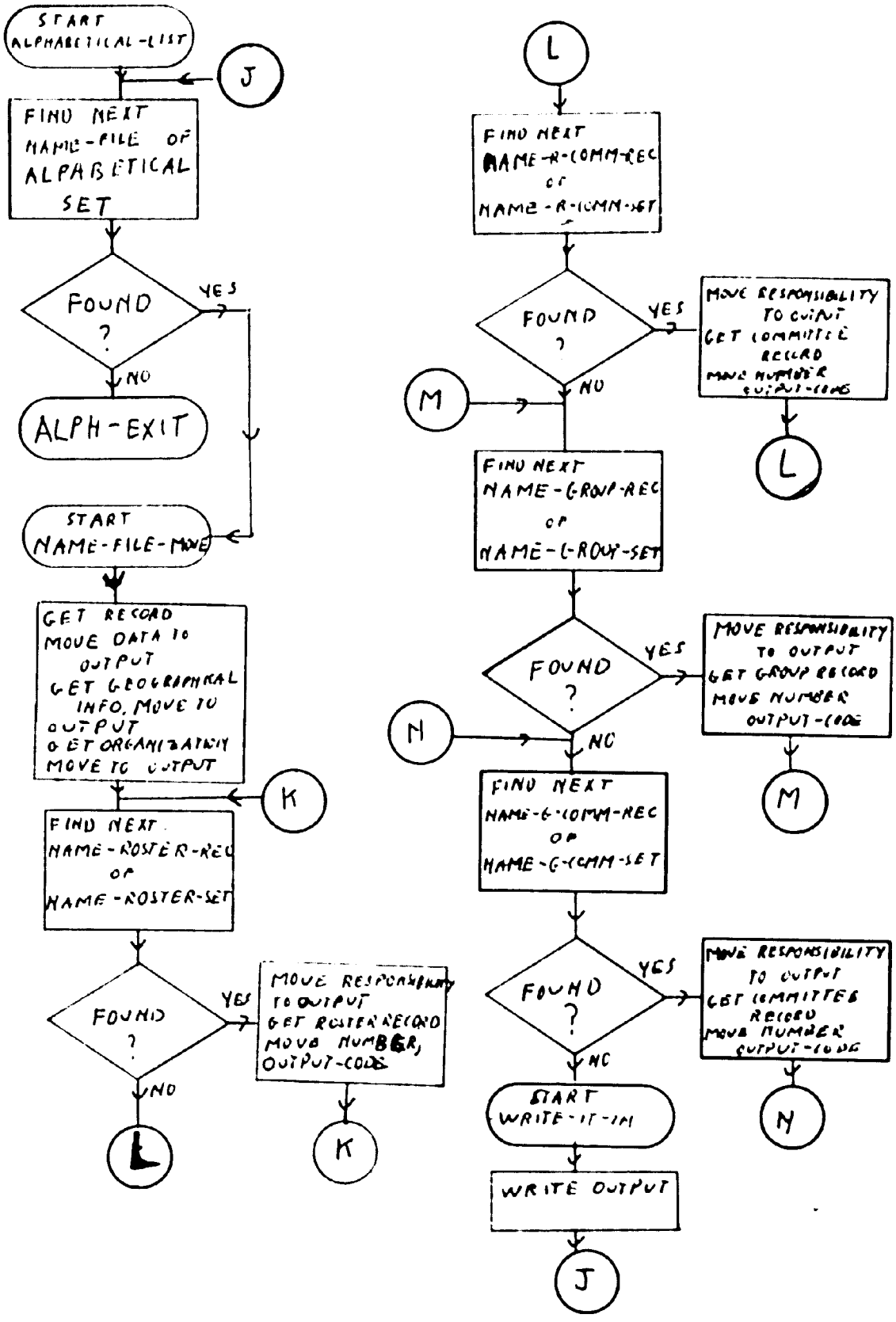
After INTERNAL-CHECK-2, the roster and the group are found with their data base keys, and processing branches back to ROSTER-WITH-GROUPS.

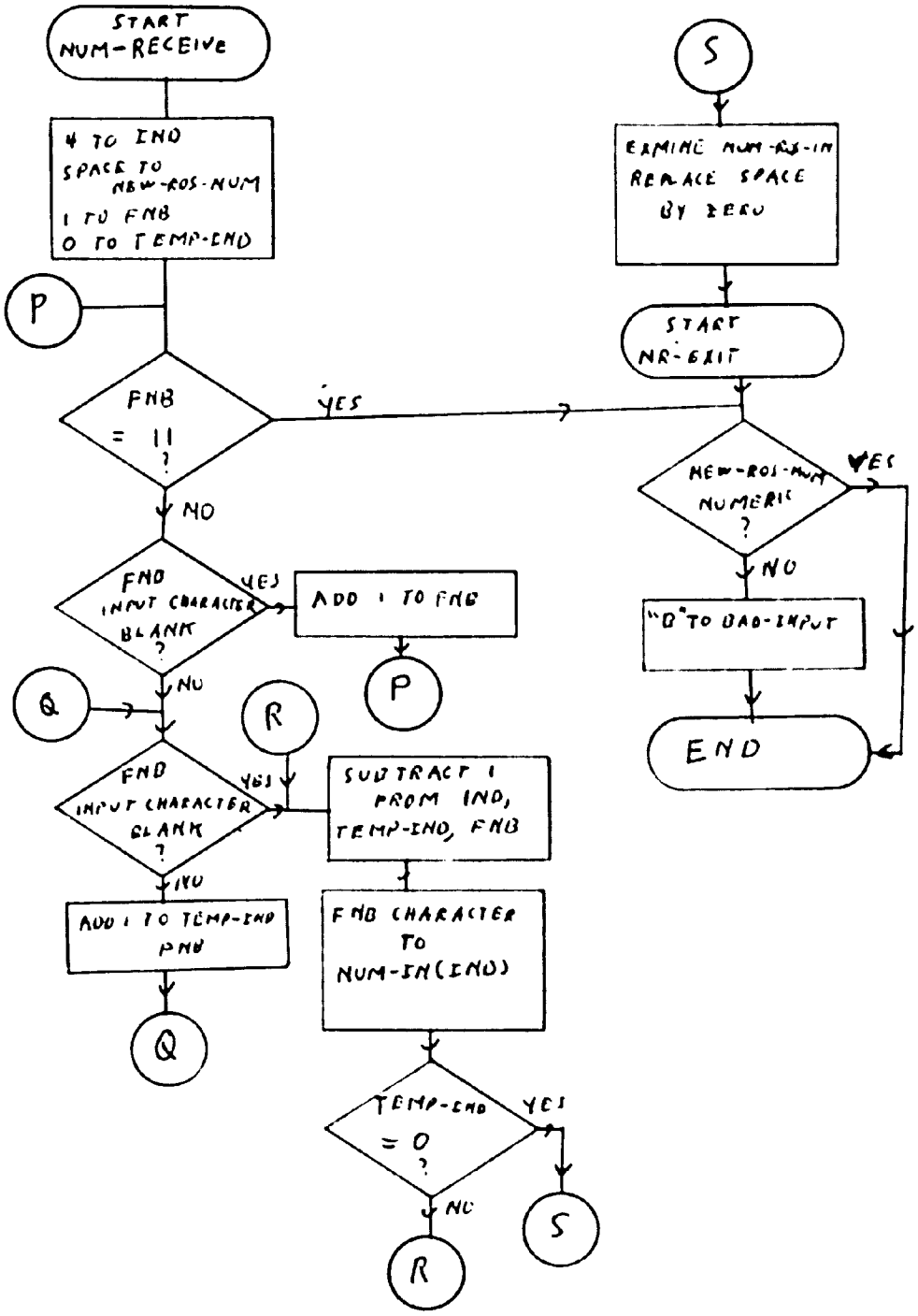

START INTERFACE PROGRAM
OPEN OUTPUT FILE
OPEN DATABASE AREAS
DISPLAY OPTIONS
ACCEPT-ROS-NUM THRU ARN-EXIT
NEW-ROS-NUM = "ABC" ?
YES
ALPHABETICAL LIST THRU ALPH-EXIT
B
NO
NEW-ROS-NUM = "ALL" ?
YES
FIND FIRST ROSTER-REC OF TALL-AREA
ROSTER-MOVE THRU ALL-EXIT
SEE ALL-ROSTERS
ALL-ROSTERS THRU ALL-EXIT
UNTIL NO MORE ROSTERS
B
NO
A
A
NEW-ROS-NUM = "ONLY-ROS" ?
YES
ONLY-ROS THRU OR-EXIT
B
NO
C
B
CLOSE FILE
CLOSE DATABASE AREAS
STOP RUN

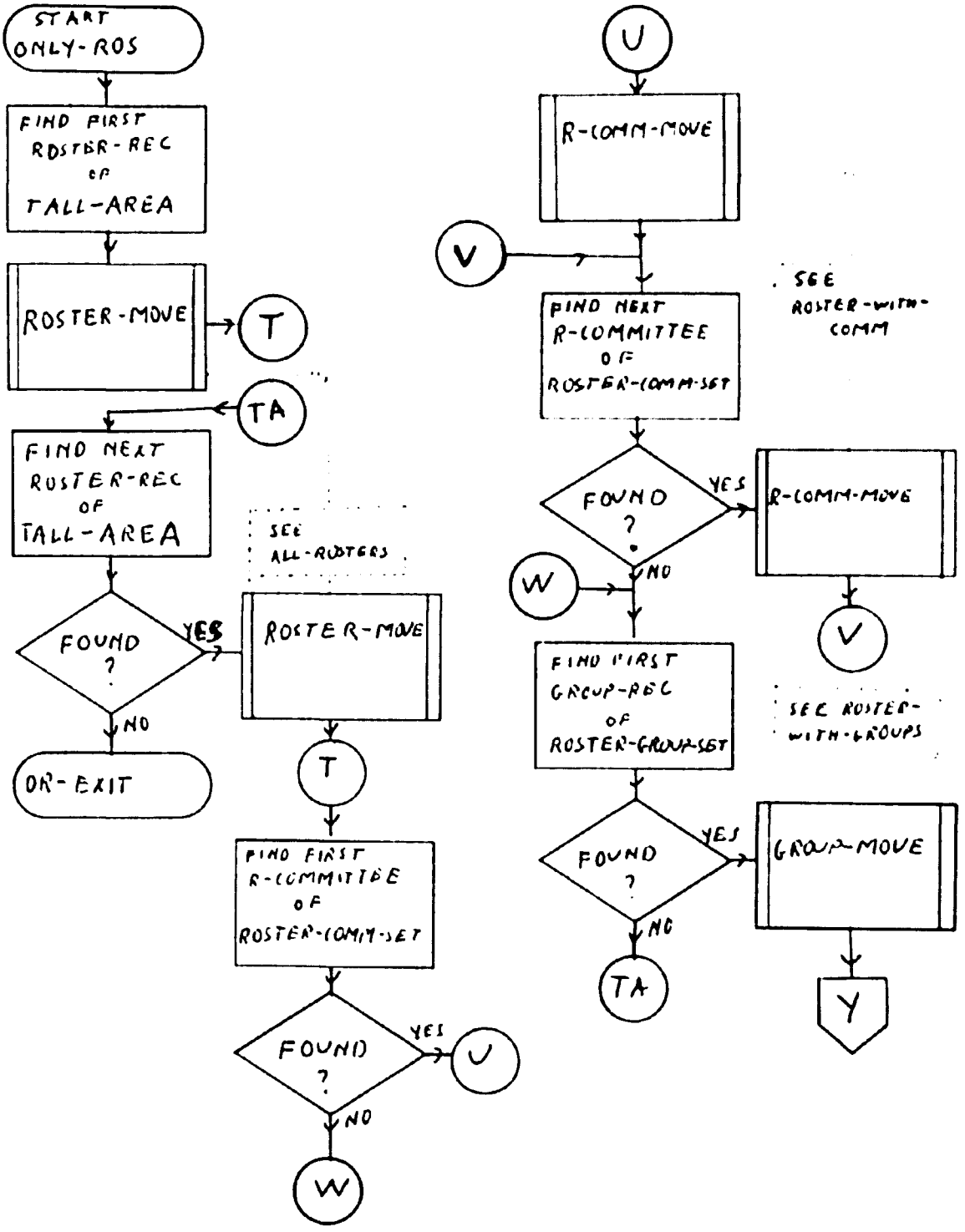

C
NUMBER OF ROSTER ?
YES
ROSTER-MOVE THRU ALL-EXIT
SEE ALL-ROSTERS
NO
ROSTER COMMITTEE ?
YES
R-COMM-MOVE THRU INTERNAL-EXIT
SEE ROSTER-WITH-COMM
NO
GROUP ?
YES
GROUP-MOVE THRU INTERNAL-CHECK-2
SEE ROSTER-WITH-GROUPS
NO
GROUP COMMITTEE ?
YES
G-COMM-MOVE THRU INTERNAL-CHECK-1
NO
MORE NUMBERS ?
YES
C
NO
B

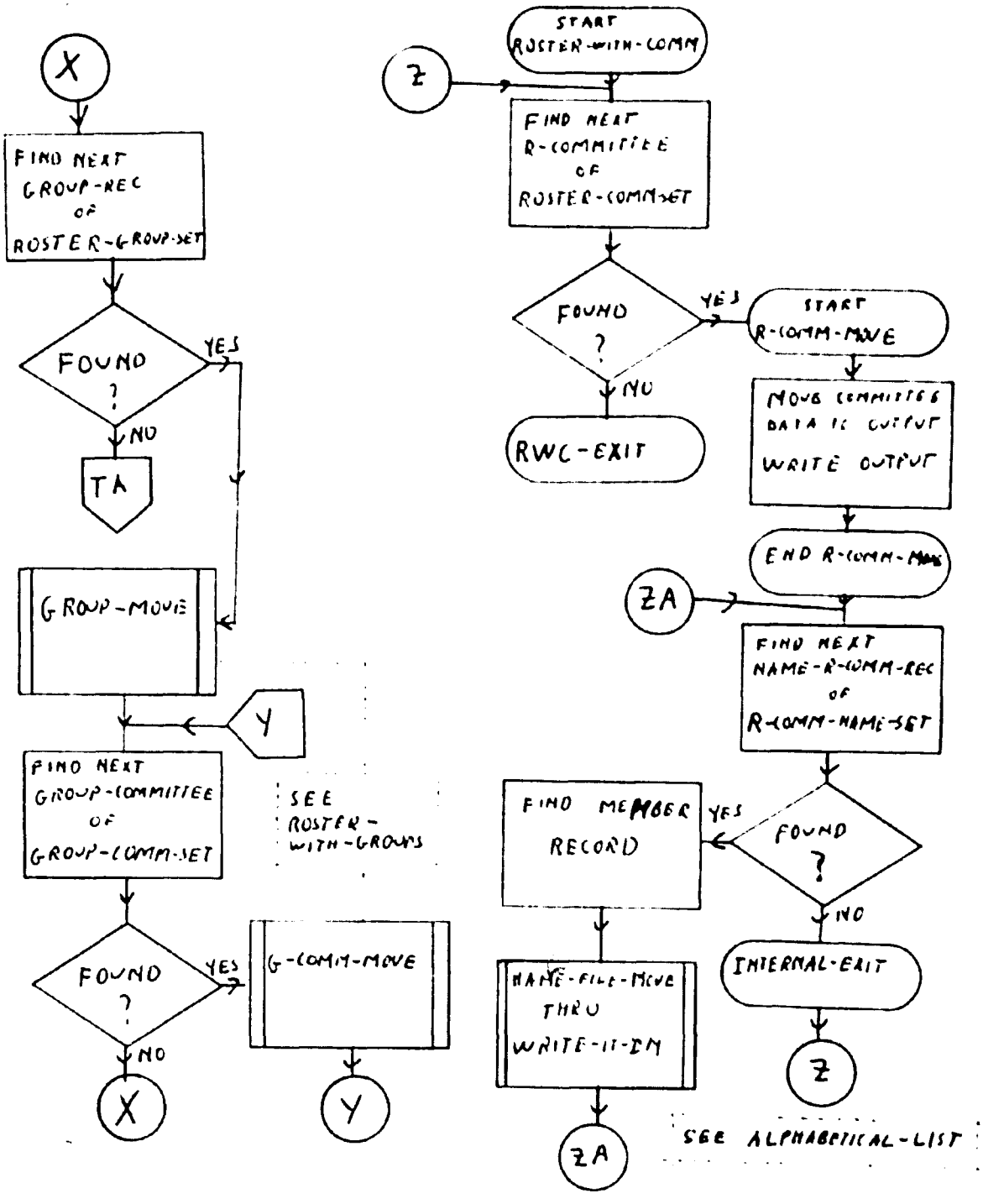

START
ACCEPT-ROS-NUM
D
DISPLAY
MESSAGE
ACCEPT
TERMINAL
INPUT
INPUT
BLANK
?
YES
ARN-EXIT
NO
NUM-RECEIVE
THRU
NR-EXIT
NEW-ROS-
NUM
= "ABC" OR
= "ALL" OR
= "ROS"
?
YES
NO
INPUT
VALID
?
YES
E
NO
DISPLAY
MESSAGE
THIRD
CHANCE
?
YES
NO
B
D
E
FIND ROSTER-REC
FOUND
?
YES
ADD 1 TO NUM-ROS-IN
KEEP
IDENTIFIERS
D
NO
FIND ROSTER-
COMMITTEE
FOUND
?
YES
ADD 1 TO
NUM-ROS-IN
KEEP
IDENTIFIERS
D
NO
FIND
GROUP-REC
FOUND
?
YES
ADD 1 TO
NUM-ROS-IN
KEEP
IDENTIFIERS
D
NO
FIND GROUP-
COMMITTEE
FOUND
?
YES
ADD 1 TO
NUM-ROS-IN
KEEP
IDENTIFIERS
D
NO
F

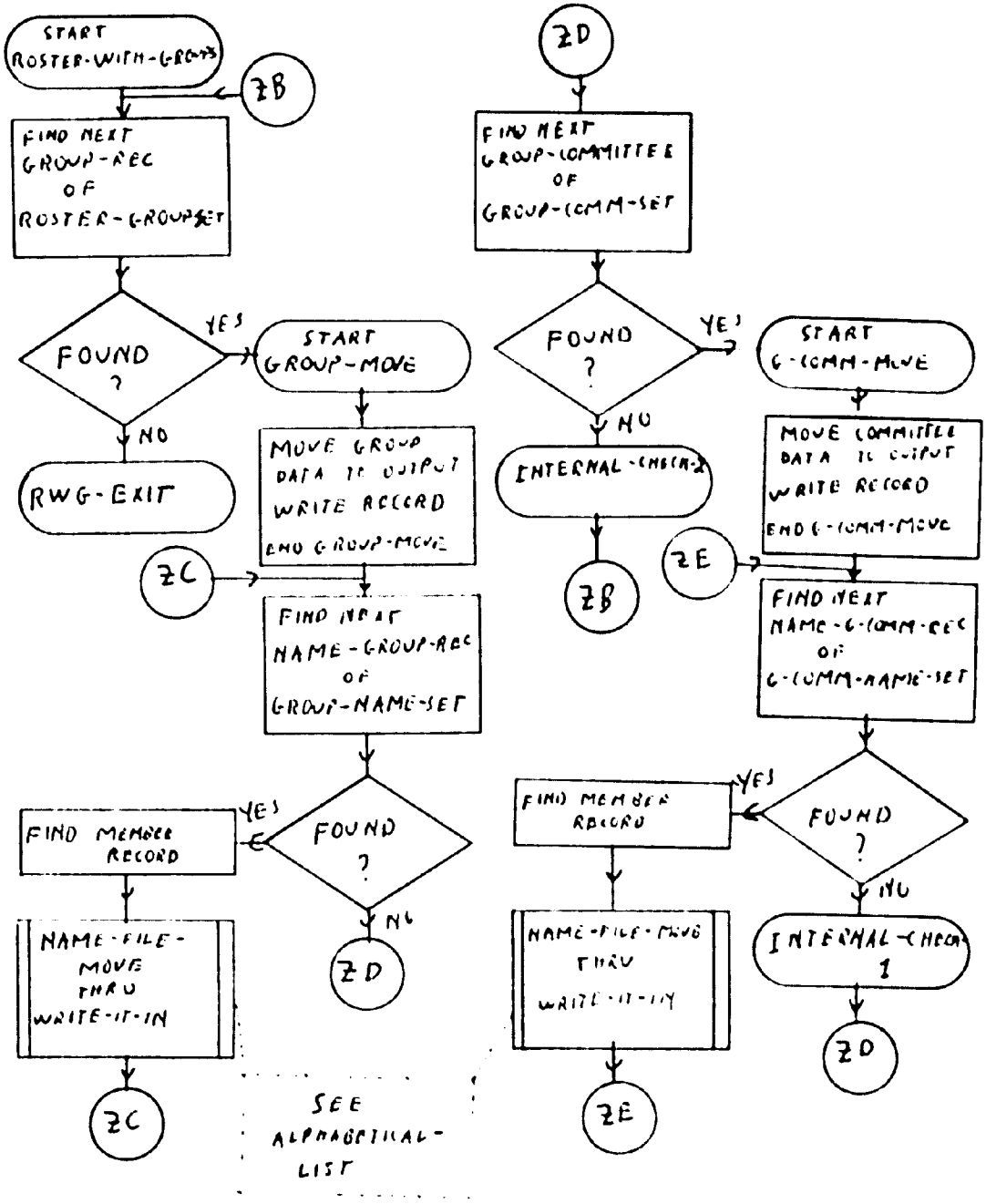

F
DISPLAY MESSAGE
THIRD CHANCE ?
YES
NO
B
D
START ALL-ROSTERS
FIND NEXT ROSTER-REC OF TALL-AREA
FOUND ?
G
1 TO DONE
I
G
START ROSTER-MOVE
DATA TO OUTPUT WRITE RECORD
END ROSTER-MOVE
ROSTER HAVE GROUPS ?
YES
ROSTER-WITH-GROUPS THRU RWG-EXIT
NO
ROSTER-WITH-COMM THRU RWC-EXIT
I
ROSTER HAVE COMMITTEES ?
YES
ROSTER-WITH-COMM THRU RWC-EXIT
I
NO
H
FIND NEXT NAME-ROSTER-REC OF ROSTER-NAME-SET
FOUND ?
YES
FIND MEMBER RECORD
NAME-FILE-MOVE THRU WRITE-IT-IN
SEE ALPHABETICAL LIST
NO
I
H
I
ALL-EXIT


START ALPHABETICAL-LIST
J
FIND NEXT NAME-FILE OF ALPABETICAL SET
FOUND ?
YES
NO
ALPH-EXIT
START NAME-FILE-MOVE
GET RECORD MOVE DATA TO OUTPUT GET GEOGRAPHICAL INFO. MOVE TO OUTPUT GET ORGANIZATION MOVE TO OUTPUT
K
FIND NEXT NAME-ROSTER-REC OR NAME-ROSTER-SET
FOUND ?
YES
MOVE RESPONSIBILITY TO OUTPUT GET ROSTER RECORD MOVE NUMBER, OUTPUT-CODE
NO
K
L
FIND NEXT NAME-R-COMM-REC OR NAME-R-COMM-SET
FOUND ?
YES
MOVE RESPONSIBILITY TO OUTPUT GET COMMITTEE RECORD MOVE NUMBER OUTPUT-CODE
NO
M
L
FIND NEXT NAME-GROUP-REC OR NAME-GROUP-SET
FOUND ?
YES
MOVE RESPONSIBILITY TO OUTPUT GET GROUP RECORD MOVE NUMBER OUTPUT-CODE
NO
N
M
FIND NEXT NAME-G-COMM-REC OR NAME-G-COMM-SET
FOUND ?
YES
MOVE RESPONSIBILITY TO OUTPUT GET COMMITTEE RECORD MOVE NUMBER OUTPUT-CODE
NO
N
START WRITE-IT-ALL
WRITE OUTPUT
J


START
NUM-RECEIVE
4 TO END
SPACE TO
NEW-ROS-NUM
1 TO FNB
0 TO TEMP-END
P
FNB
= 11
?
YES
NO
FNB
INPUT CHARACTER
BLANK
?
YES
ADD 1 TO FNB
P
NO
Q
R
FNB
INPUT CHARACTER
BLANK
?
YES
SUBTRACT 1
FROM IND,
TEMP-IND, FNB
NO
ADD 1 TO TEMP-IND
FNB
Q
FNB CHARACTER
TO
NUM-IN(IND)
TEMP-IND
= 0
?
YES
NO
S
R
S
EXAMINE NUM-RS-IN
REPLACE SPACE
BY ZERO
START
NR-EXIT
NEW-ROS-NUM
NUMERIC
?
YES
NO
"0" TO BAD-INPUT
END


START
ONLY-ROS
FIND FIRST
ROSTER-REC
OF
TALL-AREA
ROSTER-MOVE
T
TA
FIND NEXT
ROSTER-REC
OF
TALL-AREA
SEE
ALL-ROSTERS
FOUND
?
YES
ROSTER-MOVE
NO
OR-EXIT
T
FIND FIRST
R-COMMITTEE
OF
ROSTER-COMM-SET
FOUND
?
YES
U
NO
W
U
R-COMM-MOVE
V
FIND NEXT
R-COMMITTEE
OF
ROSTER-COMM-SET
SEE
ROSTER-WITH-
COMM
FOUND
?
YES
R-COMM-MOVE
NO
W
V
FIND FIRST
GROUP-REC
OF
ROSTER-GROUP-SET
SEE ROSTER-
WITH-GROUPS
FOUND
?
YES
GROUP-MOVE
NO
TA
Y


X
FIND NEXT GROUP-REC OF ROSTER-GROUP-SET
FOUND ?
YES
NO
TA
GROUP-MOVE
Y
FIND NEXT GROUP-COMMITTEE OF GROUP-COMM-SET
SEE ROSTER-WITH-GROUPS
FOUND ?
YES
G-COMM-MOVE
NO
X
Y
Z
START ROSTER-WITH-COMM
FIND NEXT R-COMMITTEE OF ROSTER-COMM-SET
FOUND ?
YES
NO
START R-COMM-MOVE
RWC-EXIT
MOVE COMMITTEE DATA TO OUTPUT
WRITE OUTPUT
END R-COMM-MOVE
ZA
FIND NEXT NAME-R-COMM-REC OF R-COMM-NAME-SET
FIND MEMBER RECORD
YES
FOUND ?
NO
NAME-FILE-MOVE THRU WRITE-IT-EM
INTERNAL-EXIT
Z
ZA
SEE ALPHABETICAL-LIST


START ROSTER-WITH-GROUPS
2B
FIND NEXT GROUP-REC OF ROSTER-GROUP-SET
FOUND ?
YES
NO
START GROUP-MOVE
RWG-EXIT
MOVE GROUP DATA TO OUTPUT
WRITE RECORD
END GROUP-MOVE
2C
FIND NEXT NAME-GROUP-REC OF GROUP-NAME-SET
FOUND ?
YES
NO
FIND MEMBER RECORD
NAME-FILE-MOVE THRU WRITE-IT-IN
2C
2D
SEE ALPHABETICAL-LIST
2D
FIND NEXT GROUP-COMMITTEE OF GROUP-COMM-SET
FOUND ?
YES
NO
INTERNAL-CHECK-2
2B
START G-COMM-MOVE
MOVE COMMITTEE DATA TO OUTPUT
WRITE RECORD
END G-COMM-MOVE
2E
FIND NEXT NAME-G-COMM-REC OF G-COMM-NAME-SET
FOUND ?
YES
NO
FIND MEMBER RECORD
NAME-FILE-MOVE THRU WRITE-IT-IN
2E
INTERNAL-CHECK-1
2D

APPENDIX G

USER MANUALS

USER MANUAL FOR THE UPDATE PROGRAM

I. INTRODUCTION

The update program allows the user to add a new record to the data base, modify an existing record, and/or delete an existing record. The additions can be for a new person member, a new organizational member, or a new roster, group, or committee. The modifications can be for any item in the record of a member, or for any item in the record of a roster, group, or committee except the field used for the internal identification number (should the user wish to change that item, it would be necessary to delete the old record and create a new one from scratch, including re-linking all the members). In addition to the above listed modifications, the user can change the organization and/or geographic lists that a member is on. The user may also alter the links to rosters, groups, and/or committees for an individual member, and/or alter the special responsibility the member has with those rosters, groups, and committees. Changes of the types mentioned in the last two sentences can only be made by accessing the record of the individual member first. The record of a member, roster, group, or committee can also be deleted

completely from the data base. The deletion of a member's record will automatically delete the name from all lists that it was on, so there is no need to separately remove the member's name from the lists.

The user can obtain information about what is currently in the data base by running the interface program (TEMPFI.CBL) and getting a printout of the created file. This will provide the user with geographical and organizational data for each member, and also a list of the rosters, groups, and committees that the member is on, together with the member's responsibility.

The user can also obtain data in two other ways. For a limited number of members, it may be preferable to use the inquiry program (READDB.CBL). For a complete dump of the data base, together with all linkages, the utility DBINFO (see the recovery manual) can be used.

The update program is written so that during execution all user input follows the prompt ==>. Whenever the prompt appears, the machine waits for user input. The user should enter the correct data, and then press the return key. If there are no values to input, just press the return key. This is equivalent to inputting all blanks. If a mistake is made before the return key is pressed, the entire entry can be remade by holding

down the CTRL key, and pressing U. The actual input starts with the first non-blank character, so spacing before an entry is not relevant, as long as the total input is not more than 50 characters.

In some cases the machine will recognize improper input, and will print a message to that effect. In such cases the user will usually have the opportunity to try again. In general, should the user enter three consecutive improper inputs, the program will cease execution. When the input must be a specific type, the machine will usually print a message explaining what is considered valid.

It is possible that during execution a data base exception will occur. If it does, a message will appear which looks like

```
STATUS/AREA/RECORD/SET = -----/.../.../...  .
```

In place of the four blanks, there will be a three or four digit number called the error-status. The dots will be replaced either by a blank or by data names used in the data base description. A particular example could be something like

```
STATUS/AREA/RECORD/SET = 322//ORGANIZATION-REC/ORGAN-NAME-SET
```

which could occur if the machine tried to find the organization of a particular member and the member had no organization listed.

An isolated exception message should not be cause for concern. However, should they appear frequently, or should the last two digits in the error-status be higher than 50, it is likely that the DBMS system has crashed. In such a case cease execution as soon as possible (try not to abort the program prematurely, however) and report it to the director of computer systems at Fritz Lab. Generally, updates made prior to a system crash will be intact, and those made after the crash will be non-existent. The update in progress during the crash could be messed up. It may be advisable to delete that record, and reenter it when system recovery has been completed.

During execution of the update program files are created on disc listing the additions, deletions, and modifications. The files are called ADDED.DAT, DELETE.DAT, and CHANGE.DAT. If a listing is desired, ask for the listing before the update program is run again, because at that point the old files will be deleted.

The program is written so that the machine will display a message about each user input. Some of the messages take several lines. If there is a large number of updates to make at a given sitting, the user may prefer to use a hard-wired CRT (there are some, for

example, at Christmas-Saucon) to decrease the time necessary for the message displays to appear, hence to speed up the overall process.

II. ADDITIONS

The user is first asked to enter P,O,R,C, or G to represent the addition of a person member, organizational member, roster, group, or committee. Any input starting with one of those characters would be acceptable. Thus, if the user were to enter PETUNIA, the program would branch to the coding used to process the record of a new person member. If any character other than those listed above is entered, the machine will display an error message.

A. NEW PERSON MEMBER

First the machine will ask for the last name of the person, then the first name, and then the middle name. The last name can be up to 20 characters long (starting with the first non-blank character), and the first and middle names can be up to 14 and 13 characters respectively. Excess characters will be ignored. Following the name, the machine will ask for the title, e.g. Mr., Miss, Ms, Dr., Prof, etc., which can be up to 17 characters long.

After the title the machine will request the address as it would appear on a four line address label. The address as given in this part is not checked for content, but merely stored as input. The city and state or country, which are used mostly for sorting purposes, and the zip code will be requested later. The zip code will be placed as the last 5 characters on the fourth line, and should not be entered until specifically requested. The second, third, and fourth lines of the address are stored as entire fields and should be entered exactly as the user wishes to see them on a label even if it means repeating the city and state or country. The second and third lines can be up to 32 characters long, and the fourth line can be up to 33 characters long (28 if space is to be reserved for the zip code).

Following the address, the computer will ask if the person is located in the United States. This is the only yes-no question which needs specific input, i.e., Y or N. If Y is entered, the computer will request the state (up to 27 characters). If N is entered, the computer will request the country (up to 32 characters). After state or country, the computer will request the city (up to 32 characters). If the state or country is not one for which the data base has an existing record

(which would probably be the case if the entry were misspelled), the computer will ask the user to reenter the name. This should be done even if the original spelling was correct. If the city is in a state or country that had previously been in the data base, then the computer will also ask for a reentry if it cannot find the city as an existing record. These are the only cases for which the program provides facilities to correct possible input errors during an update. It is done here to avoid cluttering up the data base with extraneous records which would never be used. In other cases errors can be corrected by using the update program.

The next items the computer asks for are the phone number, the starting date, and the organization of the member. The display explains how to enter the phone number and starting date. The organization name can be up to 34 characters.

Finally the computer will ask for the rosters that the person should be linked to. This means rosters, committees, and groups. Before it does so, there might be a slight delay, as the machine has some processing to do. The delay should not be more than 2 or 3 seconds. Then the user should enter the identifying number (up to three digits) of each entity as the machine requests them. The rosters, groups, and committees must have

been previously entered into the data base. After the member is linked to each roster, group, or committee, the computer will request the member's responsibility within that entity. If there is no special responsibility such as chairman, editor, etc., this entry may be left blank. Up to 6 characters may be entered, but only the first one will be used in many applications. When the last connection is made, the user should enter blanks as a signal that the inputs are finished.

B. NEW ORGANIZATIONAL MEMBER

The computer starts by asking for the name of the organization. The name should be entered as one line using up to 34 characters. Then, starting with questions about the address, the computer will ask the same questions as it does for a new person member, except that it will not ask for a title or an organization.

C. NEW ROSTER, GROUP, OR COMMITTEE

The computer will start by asking for the number of the roster or committee. This actually means roster, group, or committee. The entry should be an integer number up to three digits, and should be unique, i.e., it should not be in use currently for any existing roster, group, or committee.

The next item the computer will ask for is the

roster or committee type. This should be a 2 character entry. The type was originally used as the first two characters of the eight character identifier used with the old system. For example, roster 20S had the type SC for steering committee.

The next request will be for the identifier. The identifier was originally used as the last six characters of the eight character field used for identification under the old system. Of course it is not necessary to use all six characters. The identifier for roster 20S would just be 20S.

Following the identifier, the machine will ask for the output code. The output code is a 4 character field which will be used to identify the roster, group, or committee on certain printed lists.

After the output code, the machine will ask for the description. The description is in two parts as explained in the machine display.

If the new entity is a roster, the machine will store the new record and then ask for the next update from the beginning. If the new entity is a group, the machine will ask for the identifying number of the roster that the group should be contained in. Again, this number is a 3 digit number unique for the roster. The roster must have its record already entered into the

data base.

If the new entity is a committee, the machine will ask for the number of the roster that contains the committee. If it is a group committee, the user may enter the number of the group that contains this committee. If the user enters the roster number, the machine will then ask for the associated group anyway.

Remember that if a new group or roster committee is added, the associated roster must have its record already in the data base. If a group committee is added, the associated group must have its record already in the data base.

III. MODIFICATIONS

The machine will start by displaying a message explaining the types of modifications which may be made. In this context the word ROSTER is used as a generic term meaning either roster group or committee. The user should enter either P (for person member), O (for organizational member), or R (for roster, group, or committee). No other input will be accepted by the machine.

A. MODIFICATIONS FOR A PERSON MEMBER

This branch of the program directs the machine to start by asking if the user wants to enter only the

last name, and have the machine search through the records of all the members with the input last name, or if the user wants to enter the entire name. If the user elects to enter only the last name, the machine will display successive names and addresses until the user indicates (by entering Y) that the correct record has been found.

Entering the entire name is quicker. If the user elects to enter the entire name, the machine will ask, in this order, for the last name, the first name, and the middle name of the person as three separate inputs. It is very important to enter the name exactly as it is recorded before the change. If the name is not entered correctly, the machine will not be able to find the correct record for modification. If the name is entered correctly, the machine will display the person's name and address and ask the user to verify that the correct record has been found. This is because it is possible for several members to have the same name. If the user indicates that the record is not the record for the correct member, the machine will look for a record of a member with the same name and repeat the above process. This will continue until either the correct record is found or there are no more records to check. In the latter case, a message to that

effect will be displayed. The user makes the asked for indication by entering either Y (or any word starting with Y) to indicate the record is correct, and anything else to indicate that the record is not correct.

Once the correct record is found, the computer will ask if the user wants to change the name. If the answer is yes, the entire new name must be entered as it is requested, even if the only change is for one part of the name, e.g., to change the first name from just an initial to the entire first name.

After the name question the computer will ask about changing the title. Then it will ask about changing the address as it appears on an address label. If the address is to be changed, the entire new address must be entered, even if a change occurs on only one line of the address. Changes to the address refer also to changes to the zip code. If the zip code is unknown, just leave it blank. The rules for entering an address during a modification are the same as the rules for entering an address during an addition.

The next three questions, in order, are about changing the phone number, changing the starting date, and entering an end date. Then the computer will ask if the state or country is to be changed. If the answer is yes, the computer asks if the new country is the

United States. If it is, the machine will ask for the name of the new state, and then ask for the name of the new city. If the new country is not the United States, the machine will ask for the name of the new country followed by the name of the new city.

If the user does not want to change the state or country, the machine will ask if the city is to be changed. In this case and in the cases in the previous paragraph, if the new geographical entity is one for which the data base does not have an existing record, the computer will ask the user to reenter the name. This is protection against an input mistake. It is used to avoid cluttering up the data base with extraneous records.

Following state, country, and/or city changes, the machine will ask if the person's organization is to be changed. Then it will consider changes in roster, group, and/or committee affiliations. Again ROSTER is used as a generic term to stand for roster, group, or committee. As stated in the display, the machine considers deletions, additions, and changes in responsibility in that order. By deletions, the machine means deleting the member's name from a roster, group, or committee list. By additions, the machine means adding the member's name to roster, group, or committee lists.

In this case the responsibility on the roster, group, or committee must also be added. A person has a change in responsibility if his or her name remains on a roster, group, or committee list, but the special responsibility within that roster, group, or committee is to be changed.

Changes in roster, group, and/or committee affiliations are handled one such entity at a time, in the order specified above. The user is asked to start each change by entering the identifying number of the entity involved. When changes of one type have been completed, a blank should be entered as a signal to go on to the next type.

B. MODIFICATIONS FOR AN ORGANIZATIONAL MEMBER

These are almost the same as modifications for a person member. The main differences are that the name is entered as a whole instead of in parts, and there are no questions about changes in title or organization. Also, since an organization cannot be a chairman, vice chairman, editor, etc., of a roster, group, or committee, there are no questions about changes in responsibility when changes in roster affiliations are considered.

C. MODIFICATIONS FOR A ROSTER, GROUP, OR COMMITTEE

The computer will first ask for the roster or committee number. This means the three digit identifying number of the roster, group, or committee. This identifying number is the only data item in the record of a roster, group, or committee which cannot be modified.

The first question asked is if the user wishes to change the type. The type is the first two characters of the eight character identifier used with the old system. For example, roster 20S had the type SC for steering committee.

Next the machine will ask if the identifier is to be changed. The identifier is the last six characters of the old identification. Of course it is not necessary to use all six characters. In roster 20S the identifier is 20S.

After the identifier, the next item considered for change is the description. Finally the output code is considered for change. The description is in two parts, the first is 32 characters, and the second is 30 characters. If there is a change, both parts must be reentered.

IV. DELETIONS

The machine starts by asking the user to enter

P, O, or R. P is entered if the record of a person member is to be deleted. O is entered if the record of an organizational member is to be deleted. R is entered if the record of a roster, group, or committee is to be deleted.

A. DELETION OF A PERSON MEMBER'S RECORD

The user is asked if he (or she) wishes to enter only the last name, or the entire name. If the user elects to enter the last name only, the machine will successively display the names and addresses of all the members with the given last name until the user indicates (by entering a Y) that the correct record has been found. If the user elects to enter the entire name (which is quicker), the machine will ask for the last name, the first name, and the middle name in that order. The name must be entered exactly as it is presently recorded in the data base. Then the machine will display the name and address of the person, and ask the user to verify that the correct record has been found. This process will continue until either the correct record is found, or there are no more records of members with the input name.

B. DELETION OF AN ORGANIZATIONAL MEMBER

The user is asked to enter the name of the

organization. Then the machine proceeds as in the case of a person member.

C. DELETION OF A ROSTER, GROUP, OR COMMITTEE

The user is asked to input the three digit identifying number of the roster, group, or committee. Then the description of that entity is displayed, and the user is asked to verify that the correct entity is being considered for deletion.

V. FINAL COMMENTS

A. USE OF THE UPDATE PROGRAM

To avoid confusion, the update program is designed so that it cannot be used if the data base is being used for any other purpose, say for the inquiry program or the interface program. That means the update program cannot access the data base until all other applications have exited from the system. Similarly, once the update program connects to the data base, no other application can access the data base until the update program makes a normal exit.

B. FAILURE TO EXECUTE

The user may find that for no discernable reason the machine will not execute a particular command (if

the system crashes, the reason is definitely discernable). A possible explanation for this is that the DEC system software (i.e. the vendor supplied program which carries out the application program commands about the data base) is still new and to a certain extent experimental. All the "bugs" have not been removed. Often difficulties can be overcome by starting the particular update over (do not re-do the parts that did execute) or even ending the execution of the update program and starting over (again, do not re-do the parts that did execute).

USER MANUAL FOR THE INQUIRY PROGRAM

The inquiry program is a program written to allow a user to interrogate the data base about the data associated with an individual member. Should the user wish a list of the members associated with a particular roster, group, or committee, the interface program (TEMPFI.CBL) can be used for that purpose. The inquiry program is designed to work interactively. It does not create any files. It cannot change anything in the data base, but can only read what is already there.

The inquiry program is written so that during execution all user input follows the prompt ==> . Whenever the prompt appears, the machine waits for user input. The user should enter the correct data, and then press the return key. If there are no values to input, just press the return key. This is equivalent to inputting all blanks. If a mistake is made before the return key is pressed, the entire entry can be remade by holding down the CTRL key, and pressing U. The actual input starts with the first non-blank character, so spacing before an entry is not relevant, as long as the total input is not more than 10 characters for one letter input, and 50 characters for name input.

The user is first asked to input P or O to indicate whether the record to be interrogated is for a person member, or for an organizational member. The user may actually enter anything starting with either of those letters, as the program is written so that it will only look at the first non-blank character of the input in this case. If anything other than P or O is entered, the machine will regard it as a signal to end the program, and will cease execution. For questions that can be answered "yes" or "no", the machine will regard any input beginning with Y as a "yes" answer, and anything else, including blanks, as a "no".

After identifying the type of member, the user will have to enter the member's name. It is very important to enter the name exactly as it is presently recorded. If the name is entered in any other form, the machine will be unable to locate the correct record. The name of a person member is entered in three parts (any of which can be blank) as the machine requests them. The last name is entered first, then the first name, and then the middle name. For an organizational member, the name is entered all at once. After the name is entered, the machine will try to find the associated record. If it cannot be found, the machine will display a message to that effect. If the name is found, the machine will

display the name and address, and ask the user to verify that the correct record has been found. This process will repeat until either the correct record has been found, or there are no more records of members with the input name. The above procedure is carried out because it is possible for several members to have the same name.

Once the correct record is found, the machine will ask, in order, if the user wants to see the geographical data, the associated organization, the phone number, the start date, the end date, and finally a list of committees that the member is on. By committees the machine means rosters, groups, or committees. In each case the user is asked to indicate whether the particular item should be displayed. When the questions are finished, the machine will ask about the next member.

USER MANUAL FOR THE INTERFACE PROGRAM

The interface program is used to create a temporary file of selected rosters, groups, and/or committees for the project application programs to work on. The created file is called ROSLIS.DAT. The user should be sure that the old temporary file is either no longer needed, or copied to another file name, because whenever the interface program is run, the old ROSLIS.DAT is automatically deleted.

User interaction with the interface program is limited to entering the roster, group, and/or committee numbers wanted in the selected list. If a committee number is input, a logical picture of the output will be

committee data
member 1 data
member 2 data
.
.
.
member n data

If a group number is input, a logical picture of the output will be

group data

member 1 data

.

.

.

member n data

committee 1 data

member 1 data

.

.

.

member n1 data

committee 2 data

member 1 data

.

.

.

member n2 data

.

.

.

committee m data

member 1 data

.

.

.

member nm data

If a roster with neither groups nor committees is entered, the output will be similar to that of a committee. If a

roster with groups is entered, the output will be

roster data

output for group 1 (see above)

output for group 2

.

.

.

output for group n

If a roster with committees is entered, the output will be roster data followed by committee output (see above) for each committee in the roster.

The user has three other options. ABC can be entered, which will produce an alphabetical list of the entire membership. Also, ALL can be entered, which will produce roster output, with members, for every roster in the data base (hence every group and committee also). ROS can be entered, which will produce just a list of rosters, groups, and committees, without members.

The machine will request each user input. Should the user entry be invalid, the machine will display an error message, and request the entry again. Three consecutive invalid inputs will cause the program to terminate.

All user input will follow the prompt ==> . User input, other than ALL, ABC, or ROS, should be an integer number up to three digits, which is the internal

identifying number of a roster, group, or committee. When all of the numbers have been entered, the user should enter a blank to indicate that user input is finished.

When all user input has been entered, it will take the machine some time to complete the actual processing. This time will be several minutes at the least, and may be as much as 30 to 60 minutes. At this writing, it is impossible to tell. Do not turn off the terminal until the machine indicates that the processing has been completed. This indication will be the following:

@EXIT

Usually the terminal will also beep.

APPENDIX H

GENERAL INFORMATION MANUAL

GENERAL INFORMATION AND RECOVERY MANUAL

I. INTRODUCTION

This manual is intended as a guide in using the DEC system data base management system (DBMS) vendor manuals. It should allow the user to use the vendor manuals to obtain information about the data base, to recover from system failures, and to be able to understand how the DBMS data manipulation language (DML) is used in application programs. This is not intended as a replacement for the vendor manuals, nor will it allow the user to make major changes in the system without being familiar with the DEC DBMS (or getting help from someone who is).

II. DATA MANIPULATION LANGUAGE

The DML essentially does nothing more than add extra verbs to a host computer language. Since all the application programs for this system were written in COBOL, this manual will consider the DML verbs as being additions to the COBOL language. Only the most common situations will be considered here. A complete description of the usage of the DML is in the DECSYSTEM's

Programmer's Procedures Manual.

The verbs can be used in any COBOL sentence (where they would make sense). The most commonly used verbs are FIND, GET, MODIFY, STORE, INSERT, REMOVE, DELETE, and MOVE STATUS. Most of the verbs can be used in several different ways.

The key to most processing is the FIND verb. All the other verbs, except STORE, make sense only when they are used in conjunction with a FIND command. STORE must also be used in conjunction with a FIND command in some cases, as will be explained shortly.

The DBMS software maintains pointers to many different CURRENT records. There is CURRENT OF RECORD, CURRENT OF RUN-UNIT, CURRENT OF AREA, and CURRENT OF SET. When the FIND verb (or the STORE verb) is used, the record it finds (or stores) is automatically made the CURRENT OF RUN-UNIT, and, unless the programmer specifically directs otherwise, the CURRENT of record and any sets and/or areas that the record is in. When the other verbs are used, they act on the CURRENT OF RUN-UNIT (with some other record, in some cases).

The FIND verb, in all its forms, does nothing except set the CURRENT pointers. If the programmer wishes to do anything with any of the data items of a record, including just read them, the GET verb must be

used to copy the record into the user work area. GET does not take the record out of the data base.

Once a record has been brought into the user work area, the programmer can manipulate the record's data items with any of the usual COBOL verbs (MOVE, ADD, etc.). These changes will be made only in the work area. In order to change items in the data base, it is necessary to do the manipulations in the user work area, and then use the MODIFY verb. It is possible to use a record name following GET or MODIFY. If the CURRENT OF RUN-UNIT does not agree with the record name, the DBMS software will cause an error status to occur. It will not GET or MODIFY any record except the CURRENT OF RUN-UNIT.

A STORE command must be followed by one or more record type names. When STORE is used, it creates a new occurrence of the specified record type(s), taking as values whatever happens to be in the appropriate parts of the user work area. It also inserts the newly stored record into any sets in which the specified record type was declared an AUTOMATIC member in the schema description. The programmer must be careful here. In order to ensure that a member record is connected to the correct owner record(s), the programmer must first FIND the owner records to make them the CURRENT OF RECORD

for their record types.

If a record type was not declared as a MANDATORY AUTOMATIC member of a particular set, a specific occurrence of the record need not be in an occurrence of the set, i.e., the record need not be connected to any owner record of the set. In order to put it in an occurrence of the set, the programmer must use the INSERT verb. Before using INSERT, the programmer must FIND the correct owner record, and then FIND the record to be inserted into the set.

The REMOVE verb performs the opposite function of INSERT. It removes a record from a set in which the record type was declared as an OPTIONAL member in the schema. Nothing happens to the record itself, only some of the pointers are changed or deleted. Before using REMOVE, the programmer must FIND the record to be removed.

DELETE removes a record from the data base. A record must be the CURRENT OF RUN-UNIT before it can be deleted with the DELETE verb. The programmer can specify a record type name after the DELETE verb just in case the CURRENT OF RUN-UNIT is the wrong record type. When that happens an error status message will be returned, and nothing will be deleted. If a record to be deleted is an owner record in any sets, all the MANDATORY members

of the sets which are owned by the record will be deleted with it, and in some cases, depending on how the programmer uses the DELETE verb, the OPTIONAL members owned by the record are also deleted.

During the processing, it is often necessary to process a collection of records of a given type. Then the programmer may use the FIND NEXT option, where NEXT can be the next record of a given type in either a set or area. What the software does is use the current record of the set or area (regardless of type) as the starting point from which to look for the next record of the desired type. If the processing statements had any FIND statements (or STORE statements) before the FIND NEXT, the current record of the set or area may not be where the programmer wants it to be. One way to avoid the problem is to save the data base key of the last record of the given type, do the processing, refind that record directly using the data base key, and then FIND NEXT. An example is as follows:

```
WORKING-STORAGE SECTION.

77      HOLD-REC                        PIC 9(10) COMP.
                                          .
                                          .
PROCEDURE DIVISION.                       .
                                          .
                                          .
                                          .
```

```
FIND NEXT ROSTER-REC RECORD OF TALL-AREA AREA.
MOVE STATUS FOR RUN-UNIT TO HOLD-REC.
.
.
.
FIND ROSTER-REC USING HOLD-REC.
FIND NEXT ROSTER-REC RECORD OF TALL-AREA AREA.
```

MOVE STATUS is the verb used to save the data base key. HOLD-REC is just a programmer declared data item in working storage. The FIND .... USING .... option is a direct addressing technique, and is very fast.

III. DATA BASE INFORMATION

The DEC system supplied utility program DBINFO allows the user to obtain information about his data base. There are various types of information, and all are described in the vendor's Administrator's Procedures Manual. The type of information which might be most useful is one which gives a complete picture of what is contained in any or all of the data base areas and/or sets. In this case there are two areas, NAIM-AREA and TALL-AREA. Suppose a picture of NAIM-AREA and the set GROUP-NAME-SET of TALL-AREA is desired. The following is an example of how to obtain that picture:

```
@R DBINFO
/SCHEMA STEVE
/SS UNIVERSAL
/SUPERSEDE DATFI
/OPEN "TALL-AREA"
/DISPLAY DATA: "GROUP-NAME-SET"
/CLOSE "TALL-AREA"
/OPEN "NAIM-AREA"
/PAGES "NAIM-AREA"
/DISPLAY DATA
/CLOSE ALL
/^C
@
```

Notes:

1. @ is the machine command mode prompt.
2. R DBINFO calls the utility program.
3. / is the machine prompt for the utility program.
4. STEVE is the name of the schema, and UNIVERSAL is the name of the subschema being used.
5. DATFI is the user defined name for the file which will contain the eventual result.
6. The OPEN statement opens the areas in protected retrieval mode. If they are specified that way in the schema, a privacy key must be included.

7. The names of areas and sets must be included in quotes if a hyphen is included in the name.

8. After the command DISPLAY DATA is given, there may be a wait before the next machine prompt appears, as the data must be written into the DATFI file.

9. The open commands set the pages for those areas until the first DISPLAY command is given. Then the PAGES command must be used to reset the pages of the data base.

10. ^C appears when the user holds down the CTRL key and presses C to indicate that the machine should return to command mode.

11. At the conclusion, all the information will be in the file DATFI.DBI.

IV. DATA BASE RESTORATION

If the system should crash for any reason during an update run, the data base areas will be left in an undefined state. In order to be returned to a usable state the data base utility program DBMEND must be used to force open the data base areas and then close them. An example is the following:

```
@R DBMEND
/SCHEMA STEVE
/FORCEOPEN "TALL-AREA":FIXEM
/FORCEOPEN "NAIM-AREA":FIXEM
/CLOSE ALL
/^C
@
```

This utility is similar to DBINFO, but is actually easier to use. FIXEM is the privacy keys for the data base areas, and must be used since DBMEND actually does change the data base.

Vita

The author was born on March 31, 1943 in Springfield, Massachusetts, the son of Dr. H. Bernard Tillman, and Mrs. Jean E. Tillman.

He graduated from Springfield Classical High School in 1961. He received a Bachelor of Science in Applied Mathematics from Brown University in 1965. He received a Ph.D. in Mathematics from Brown University in 1970.

Since 1970, he has been on the faculty of Wilkes College. Currently he is an Associate Professor of Mathematics and Computer Science.